INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tel (1) 39 63 55 11

Rapports de Recherche

1992

10ème anniversaire

N° 1748

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

INTERACTION SYSTEMS I THE THEORY OF OPTIMAL REDUCTIONS

Andrea ASPERTI
Cosimo LANEVE

Septembre 1992

RR_1748

INTERACTION SYSTEMS I

The theory of optimal reductions

Andrea Asperti[1] Cosimo Laneve[2]

September 10, 1992

[1]INRIA-Rocquencourt, 78153, France. asperti@margaux.inria.fr

[2]Dpt. di Informatica, Corso Italia 40, I - 56125 Pisa, Italy. laneve@di.unipi.it

Abstract

A new class of higher order rewriting systems, called Interaction Systems, is introduced. From one side, Interaction Systems provide the intuitionistic generalization of Lafont's Interaction Nets [Laf90] (recall that Interaction Nets are *linear*). In particular, we keep the idea of binary interaction, and the syntactical bipartition of operators into *constructors* and *destructors*. From the other side, Interaction Systems are the subsystem of Klop's Combinatory Reduction Systems [Kl80, Ac78] where the Curry-Howard analogy still "makes sense". This means that we can associate with every IS a suitable logical (intuitionistic) system: constructors and destructors respectively correspond to right and left introduction rules, interaction is cut, and computation is cut-elimination.

Interaction Systems have been primarily motivated by the necessity of extending Lamping's optimal graph reduction technique for the λ-calculus [Lam90, GAL92] to other computational constructs than just β-reduction. This implementation style can be smootly generalized to arbitrary IS's, providing in this way a uniform description of essential rules such as conditionals and recursion.

The optimal implementation of IS's will be only sketched here (it will eventually be the subject of the forthcoming Part II). The main aim of this paper is to introduce this new class of Systems, to discuss the motivations behind its definition, and to investigate the *theoretical* aspect of optimal reductions (in particular, the notion of *redex-family*).

On introduit une nouvelle classe de systèmes de réécriture d'ordre supérieur: les Systèmes d'Interaction. D'une part, les Systèmes d'Interaction sont la généralisation intuitioniste des Réseaux d'Interaction de Lafont (les Réseaux d'Interaction sont linéaires); en particulier, on reprend l'idée d'interaction binaire, et la bipartition syntaxique des opérateurs en constructeurs *et* destructeurs. *D'autre part, les Systèmes d'Interaction sont la sous-classe des Combinatory Reduction Systems de Klop où on peut toujours appliquer l'analogie de Curry-Howard, c'est-à-dire qu'on peut définir, pour chaque IS, le système logique intuitioniste correspondant: les constructeurs et les destructeurs correspondent respectivement à des règles d'introduction à droite et à gauche, l'interaction à une coupure, et la réduction est l'élimination des coupures.*

Les Systèmes d'Interaction ont leur motivation primordiale dans la nécessité d'étendre la technique d'évaluation optimale pour le lambda calcul de Lamping à d'autres opérations que la β-réduction. Ce style d'implémentation peut être appliqué sans problèmes à tous les Systèmes d'Interaction, ce qui donne une description uniforme de règles de réécriture essentielles, comme les sauts conditionnels ou la récursion.

L'implémentation optimale des IS's sera seulement esquissée ici (il sera le sujet du deuxième partie). L'objectif principal de ce papier est d'introduire cette nouvelle classe des Systèmes, de motiver leur définition, et d'étudier les aspects théoriques *des réductions optimales (en particulier, la notion de* famille de radicaux*).*

1 Introduction

In 1978, Lévy introduced the notion of "redex family" in the λ-calculus. The idea was to capture the phenomenom of duplication of redexes, providing an abstract definition of what could be considered as an optimal implementation of the λ-calculus. Intuitively, since redexes in a same family are "copies" of a same redex (they have been generated in the same way), it should be possible to share their reduction.

For a long time, no implementation was able to achieve the theoretical performance fixed by Lévy (see [Fi90]), and it is only in recent years [Lam90, Ka90] that this problem has been finally solved.

The graph reduction technique proposed by Lamping, and successively, remarkably simplified in [GAL92], seems particularly promising for an actual implementation.

In particular, this implementation technique has put in evidence a restricted set of operators which provide an "optimal" sharing of common subexpressions. This fact suggested the possibility to extend the approach to a wider range of rewriting systems, including the λ-calculus as a particular case.

The class of systems we were looking for became apparent to us after Gonthier's work [GAL92] about the relations between Lamping's graphs and the geometry of interaction of Linear Logic [Gi86, Gi88]. Roughly, [GAL92] establishes a correspondance between sharing operators and the notion of box in Linear Logic (sharing operators provide a "local" implementation of boxes [GAL92, As91]). This result seemed to allow a smooth generalization of Lamping's evaluation style to a very large class of systems, by just replacing the "linear" part of the λ-calculus (i.e., the linear lambda calculus), with an arbitrary linear calculus.

The obvious candidate was provided by Lafont's Interaction Nets [Laf90].

Following the analogy

$$\lambda\text{-calculus} = \text{linear } \lambda\text{-calculus} + \text{sharing}$$

our first concern was to define the formal system corresponding to Interaction Nets + sharing. This non-linear (intuitionistic) generalization of Interaction Nets are ours Interaction Systems (IS's, for short).

In particular, in Interaction Systems, we keep the idea of binary interaction, and the syntactical bipartition of operators into *constructors* and *destructors*. We only remove the linear constraint, and add some further conditions imposed by the intuitionistic (functional) framework.

Remark 1.1 Switching from "nets" to "systems" also means for us to pass from a graph-oriented notation to the realm of rewriting systems. Luckily, Interaction Systems have an elegant and reasonably simple concrete syntax, due to their intuitionistic nature. ■

Interaction Systems turn out to be a subclass of Klop's Combinatory Reduction Systems [Kl80] (see also [Ac78]). Roughly, they are the subsystems of CRS's where the Curry-Howard analogy still "makes sense". This means that we can associate to every IS a suitable "logical" (intuitionistic) system: constructors and destructors respectively correspond to right and left introduction rules, interaction is cut, and computation is cut-elimination. As the optimal implementation in [GAL92] has been inspired

by Linear Logic [Gi88] (see also [GAL92]), this relation between IS's and Intuitionistic Logic turns out to be essential in the development of our work.

More specifically, we get Interaction Systems from CRS's by imposing a suitable condition on the shape of the left hand side of the rules, reflecting the idea of binary interaction inherent to the logical understanding of Lafont's work. This means that every reduction comes from the interaction of two operators, a destructor $\mathbf{d}$ and a constructor $\mathbf{c}$.

The lhs of the rewriting rules in Interaction Systems have thus the simple shape

$$\mathbf{d}(\mathbf{c}(\vec{x}^{\,1}.X_1,\cdots,\vec{x}^{\,m}.X_m),\ldots,\vec{y}^{\,n}.X_n).$$

The rhs is every *closed* meta-expression built up with operators of the signature, the *distinct* metavariables in the lhs, and a meta-operation of substitution (see section 2 for the formal definition).

For instance, the β-rule of the λ-calculus is expressed as follows:

$$@(\lambda(\langle x\rangle.X),Y)\ \rightarrow\ X[^{Y}/_{x}].$$

Although Interaction Systems have been primarily suggested by the need of extending Lamping's optimal graph reduction technique to other constructs than β-reduction, we shall merely sketch the solution in the present paper (an exhaustive study of the implementation will eventually be the subject of the forthcoming second part). The problem is that before coping with implementation there is a lot of theoretical work to be developed. In particular, since our aim is that of providing an "optimal" implementation, we must start with formalizing the notion of optimality. This means that all the work about Levy's families of redexes in the λ-calculus [Le78, Le80] must be suitably generalized to IS's, and this generalization, as we shall see, is much less evident of what one could imagine.

The structure of the paper is the following. We shall start with a detailed discussion about the relations between Interaction Systems, Interaction Nets and Combinatory Reduction Systems. In this section we shall also provide the main motivations, both theorical and operational, behind the introduction of this new class of higher order systems. Section 3 is devoted to the formal definition of Interaction Systems, and some other standard notions and results (residuals, standardization, and so on). Section 4 is a brief discussion of the notion of *redex-family*. Here we have a first surprising result: Lévy's *zig-zag* [Le78] is not adequate for IS's (and thus for CRS's). This proves that, despite of its abstract nature, this definition of the family relation is not general enough. Similar problems arise with the generalization of the *extraction relation* [Le80], that essentially express the causal dependencies of a redex. Only labelling passes quite smoothly to IS's, and this will be defined in section 5. In section 6 we introduce a subclass of IS's: the λ-Discrete Interaction Systems. This class is essentially our correspective of Klop's λ-TRS's. The main point of λ-DIS's is that, in this particular case, we are still able to recover the nice correspondance between labelling and the other notions of family (sections 7 and 8). This is due to the particularly simple interplay between creation of new operators and substitution in λ-DIS's. In the final section, we shall briefly sketch the optimal implementation of IS's following Lamping's graph reduction technique. However, we shall not address, in this paper, the complex proofs of correctness and optimality.

Apart from the definition of $IS's$, which is interesting in itself, and the definition of the optimal implementation (whose correctness is not proved here), the most relevant contribution of the paper is

in appendix B. This appendix is devoted to the investigation of the *structure* of labels (for IS's), and provide the base for our original proof of the equivalence between extraction and labelling in the case of λ-DIS.

2 Interaction Systems, Interaction Nets and CRS's

In this introductory section, we shall discuss the main motivations behind the definition of Interaction Systems, relating them to Lafont's Interaction Nets [Laf90] and Klop's Combinatory Reduction Systems [Kl80]. We shall particularly stress the logical (intuitionistic) nature of IS's, and the possibility to extend the Curry-Howard analogy to this class of system. Moreover, we shall introduce the problem of optimal implementation, which provided the original motivation to the definition of IS's.

2.1 From Interaction Nets to Interaction Systems

Interaction Nets have been defined by Lafont in [Laf90]. They generalize Girard's Proof Nets of Linear Logic, with the intention to provide a language whose programming discipline is "already a logic".

Instead to recall the formal definition of Interaction Nets, we shall proceed by discussing a simple example (Lafont's paper is very clear and pleasant; we strongly reccomend the reading). In particular we shall try to emphasize the logical nature of Interaction Nets (a theme that, for lack of space, has been a bit overlooked in [Laf90]).

In the following we assume that the reader has some familiarity with the Curry-Howard analogy, Gentzen's Calculus of Sequents, Proof Nets and so on.

The example of Interaction Nets we shall discuss are those describing the linear λ-calculus. The choice is somewhat obliged; the λ-calculus is a paradigmatic example of higher order rewriting system, while Interaction Nets impose to work in a linear setting. Moreover, the Interaction Nets describing the linear λ-calculus are essentially the *Proof Nets* of Linear Logic [Gi88], that started all the work.

The rules for introducing application and λ-abstraction are respectively:

$$\frac{\Delta \vdash t : A \qquad x : B, \Gamma \vdash t' : C}{\Delta, y : A \multimap B, \Gamma \vdash t'[^{@(y,t)}/_{x}] : C} \qquad\qquad \frac{\Delta, x : A \vdash t : B}{\Delta \vdash \lambda(\langle x \rangle . t) \; : \; A \multimap B}$$

In the notation of Interaction Nets, a proof of a sequent $A_1, \ldots, A_n \vdash B$ is represented as a graph with $n+1$ conclusions, n with a negative type (the inputs) and one with a positive type (the output). For instance an axiom is just a line with one input and one output. The application of the two rules above is then represented in Figure 1. Every logical rule is thus represented by introducing a new agent in the net (a new labelled node in the graph). The agent has a *main port*, individuated by an arrow, that is associated with the *principal formula* of the logical inference. The main port may be either an input or an output. In the first case it corresponds to a new logical assumption in the left hand side of the sequent, and it is called a *destructor*; in the second case it corresponds to the right hand side of the sequent, and it is called a *constructor*. In the example above, @ is a destructor, while λ is a constructor. Thus, destructors and constructors are respectively associated with left and right introduction rules of logical operators.

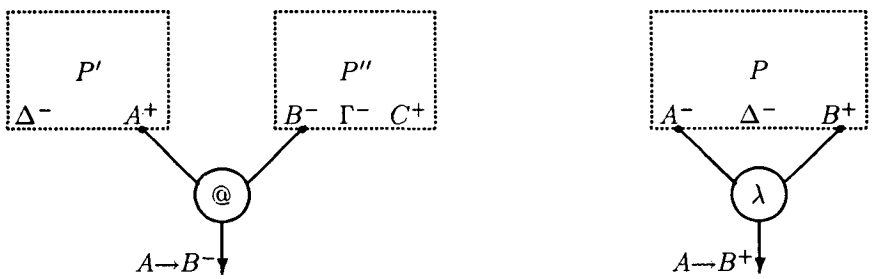


Figure 1: The graphical representation of @ and λ.

The other ports of the agents are associated with the so called *auxiliary* formulae of the inference rule, that is the distinguished occurrences of formulae which occur in the upper sequents of the rule. In the two rules above, the auxiliary formulae are A and B.

The auxiliary ports of an agent may be either inputs or outputs, independently from the fact that it is a constructor or a destructor. Actually, in the general theory of Interaction Nets due to Lafont, which is inspired by *classical* (linear) logic, there is a complete simmetry between constructors and destructors, and no restriction at all is imposed on the type of the auxiliary ports (in other words, there is a complete simmetry between inputs and outputs). On the contrary, the fact of limiting ouselves to the intuitionistic case, imposes some obvious "functional" constraint.

Note first that auxiliary formulae may "come" from different upper sequents or from a single one. For instance, in the example above, the auxiliary ports of @ are in different upper sequents, while those of λ both comes from the same upper sequent. Lafont expresses this fact by defining *partitions* of auxiliary ports. So, in the case of @, A and B are in different partitions, while in the case of λ they are in the same partition.

Remember now that the type of an auxiliary port is the opposite of the type of the conclusion it has to be matched against. Than, the intuitionistic framework imposes the following constraints:

- In every partition there is at most one negative port. If a negative port exists, we shall call it an *input* partition; otherwise it is an *output* partition.
- Every agent has exactly "one output" (functionality). In particular, if the agent is a constructor, the main port is already an output, and all the partitions must be input. Conversely, in the case of destructors, we have exactly one output partition among the auxiliary ports, and this partition has to be a singleton.

Remark 2.1 1. As the reader has surely understood, the partitions of the auxiliary ports are meant to express a binding mechanism. In case all partitions are singletons (*discrete* case, in [Laf90]) the net is acyclic, corresponding to a first order rewriting system.

2. The requirement that output classes should be singletons is not strictly nedeed from the intuitionistic point of view, in the sense that every (proof of a) *sequent* still has a functional interpretation. However we may assume a stronger point of view, namely to regard every *agent* as a functional (single output) operator. Since this approach remarkably simplifies the concrete syntax, we have finally adopted it.

   Note that the problem only concerns destructors, since the output of constructors is the main port, that is always unique. The main example of agent which is captured by assuming the "weak" constraint (one output partition) but not from our "strong" constraint (one *singleton* output partition) is the "contraction" operator (i.e. the operator of duplication). Indeed, it is a destructor with a single output partition $+,+$. Another example is provided by the operator of "dereliction" (erasing), whose output partition is empty. Since we are going to add these "structural" operators as an integrated part of our systems, the loss of generality is largely compensated by the gain in clarity.

■

2.2 Reduction Rules

The main property of Interaction Nets is the principle of binary interaction, that is the fact that agents may only interact through their principal ports. Since a net must be well typed (inputs may be only connected to outputs), this implies that we may have interaction only between a constructor and a destructor.

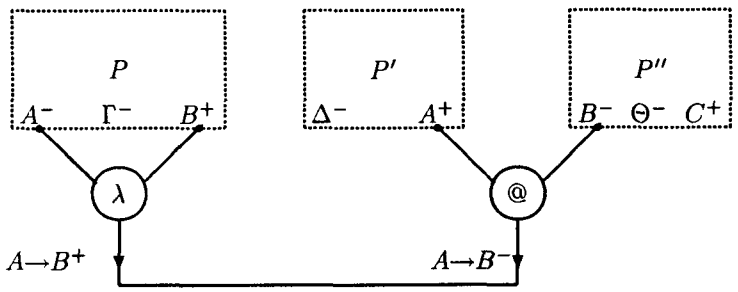
This form of interaction is a generalization of cut elimination. Let us consider again the example above. The following cut

$$\dfrac{\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \dfrac{\Delta \vdash A \qquad B, \Theta \vdash C}{\Delta, A \multimap B, \Theta \vdash C}}{\Gamma, \Delta, \Theta \vdash C}$$

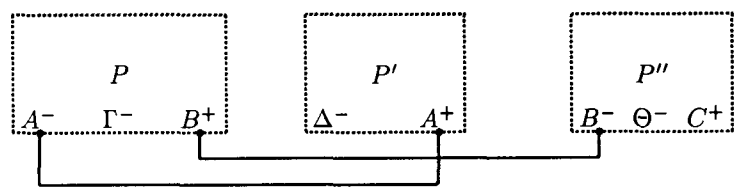
is eliminated by introducing two cuts of lesser grade, namely:

$$\dfrac{\Delta \vdash A \qquad \dfrac{\Gamma, A \vdash B \qquad B, \Theta \vdash C}{\Gamma, A, \Theta \vdash C}}{\Gamma, \Delta, \Theta \vdash C}$$

In terms of Interaction Nets, the previous situation is described by means of the following interaction between λ and @:



reduces to



Interaction Nets are *linear*. The variables in the left hand side of the rewriting rules correspond to ports in the graph and these ports are preserved by the reduction. For instance, the interaction rule between @ and λ involves four ports (those associated with the types A^-, B^+, A^+, B^- in the example above). Variables are thus an interface between the redex and the rest of the net; the only effect of the reduction rule is to change the connections of ports in the interface, possibly by introducing new operators.

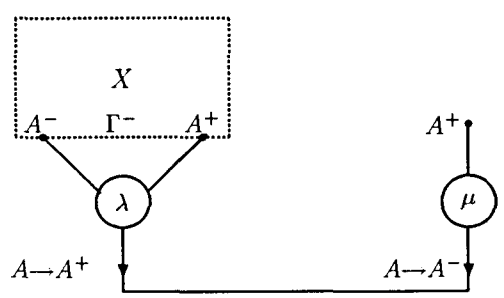
The main difference of Interaction System w.r.t. Interaction Nets is to remove this linear constraint.

Let us consider an example. An interesting destructor for the operator of λ-abstraction is the recursion operator μ. The destructor μ interacts with λ according to the following rewriting rule (see next section for more details on the concrete syntax):

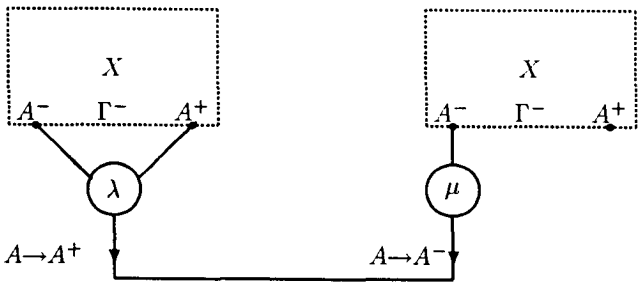
$$\mu(\lambda((x).X)) \rightarrow X[^{\mu(\lambda((x).X))}/_x]$$

Note that in the right hand side we use two times the auxiliary port of λ; so the rule is not linear.

This fact introduces several problems, wich can be simply understood by considering the graphical representation of the rule. Intuitively, we would expect to have something of the following kind:



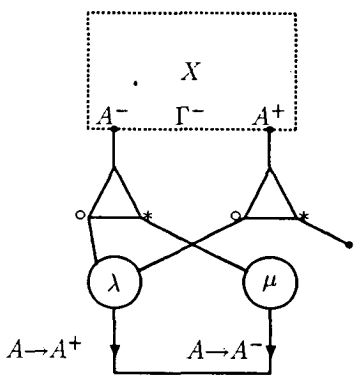
reduces to



The part of graph corresponding to X has to be duplicated. Moreover, all the free variables in M (those marked with Γ in the picture above) must be suitably shared. This means that the subterm X has to be considered as a global entity (a *box* in Girard's notation), and we loose the possibility to express the rewriting rule as a local operation interacting with the rest of the world only through an interface.

The problem is that of handling sharing of common subexpression, and the main idea of the present work is that we can safely use Lamping's operators for this purpose. This is particularly appealing, since they offer a *completely local* handling of sharing; in other words, we may still implement Interaction Systems as Interaction Nets (and providing an *optimal* implementation!).

For instance, in the example above, the intuition is to write the right hand side of the rewriting rule for the interaction between μ and λ as something of the following kind:

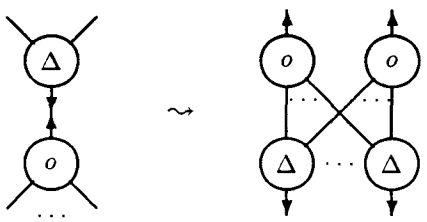


As usual, the great problem will be to introduce in the correct way the control operators (square brackets and croissants) which ensure the right matching of fan-in and fan-out.

2.3 Sharing in Interaction Nets

Interaction Nets are linear. This is not a limitation from the point of view of computability, since Interaction Nets are Turing Complete. However it is a somewhat unbearable limitation from the point of view of the linguistic expressiveness of the formal system. Lafont seems to suggest that we can solve the problem by explicitly introducing "ad hoc" duplication and erasing operators in the syntax. For instance he consider the example of multiplication, where we would write something of the form $\mathbf{Mult}(\mathbf{S}(X), Y) \rightarrow \mathbf{Add}(Y, \mathbf{Mult}(X, Y))$. This rule is not linear in X, but we may take care of the

double occurrence of X by introducing an explicit operator of duplication Δ. Δ is a destructor which has exactly the same formal status of every other operator in the system. So, in particular, we must explicitly define the interaction between Δ and every other constructor of the system; moreover, if we are in a typed setting, we have to introduce a different operator of duplication for every type. Now, it is more or less evident that duplication is a completely polymorphic operation, and in a very strong sense: indeed the behaviour of Δ is uniform w.r.t. all constructors, and it is explicitly defined by the following interaction:



That is, the operator interacting with Δ is duplicated, and we introduce as many operators of duplication as the number of auxiliary ports of o.

An intuitive way to understand the present paper, is that of providing a completely abstract approach to the problem of duplication and erasing in Interaction Nets (and, more generally to the problem of *sharing*).

This is more or less trivial in the discrete case, but things become much more entangled as soon as we have cycles.

So, an alternative way to understand our results about Interaction Systems, is that of proving that we can assure (optimal) sharing in (intuitionistic) Interaction Nets by means of Lamping's operators. (as the reader has already understood, Δ is just a Fan-in).

In particular, the set of rules in [GAL92] really looks as an abstract framework for describing a sharing mechanism; the interface between these rules and the linear part of the calculus (i.e. constructors and destructors) is then provided by rewriting rules of the form described in the above picture (see the section 9 for more details).

Remark 2.2 As far as we are in the discrete case, a duplication operator may never appear in front of the main port of a destructor. This means that destructors (and thus redexes!) are never duplicated, and sharing is trivially optimal (see next section). ■

2.4 CRS and Interaction Systems

In the previous sections we have introduced Interaction Systems as a generalization of Interaction Nets. There exists however another possible approach to Interaction Systems, namely as a subclass of Combinatory Reduction Systems.

From this respect, the main point of Interaction System is their close relation with Intuitionistic Logic; in a sense, Interaction Systems generalize the Curry-Howard analogy to logical calculi other

than the λ-calculus. The main point of this analogy is the correspondence between redexes and cuts. A computation is thus the process of cut-elimination.

In case of intuitionistic systems, the cut-rule looks as follows:

$$\frac{\Delta \vdash A \quad A, \Gamma \vdash B}{\Delta, \Gamma \vdash B}$$

The only interesting case in the cut-elimination process is when the cut-formula A has been just introduced in both the upper sequents of the cut-rule. In this case, the cut may be regarded as a binary interaction between a constructor (right introduction rule) and a destructor (left introduction rule). This is the main reason for introducing two classes of operators, and to restrict the interaction to the binary case.

The best way to understand an Interaction System as a rewriting system, is to start with considering the discrete case, which is particularly simple. In the discrete case we have no binders, so we are at a first order level. The signature of the system is a first order signature Σ partitioned in two classes: the constructors (ranged over by $\mathbf{c}$) and the destructors (ranged over by $\mathbf{d}$).

The rewriting rules have the following general shape:

$$\mathbf{d}(\mathbf{c}(X_1, \cdots, X_m), \cdots, X_n) \rightarrow t$$

where $t \in T_\Sigma(X_1, \cdots, X_n)$.

Example 2.3 A typical example of Interaction System is *Primitive Recursion*. We have only two constructors 0 and $\mathbf{S}(-)$. Composition of two functions $\mathbf{f}(-)$ and $\mathbf{g}(-)$ is obviously defined as $\mathbf{f}(\mathbf{g}(-))$, and definition by primitive recursion has already the right form, indeed:

$$\begin{aligned} \mathbf{d}(0, X) &\rightarrow \mathbf{h}(X) \\ \mathbf{d}(\mathbf{S}(X), Y) &\rightarrow \mathbf{f}(X, Y, \mathbf{g}(X, Y)) \end{aligned}$$

For instance, we may define

$$\begin{aligned} \mathbf{Add}(0, X) &\rightarrow X & \qquad \mathbf{Mult}(0, X) &\rightarrow 0 \\ \mathbf{Add}(\mathbf{S}(X), Y) &\rightarrow \mathbf{S}(\mathbf{Add}(X, Y)) & \qquad \mathbf{Mult}(\mathbf{S}(X), Y) &\rightarrow \mathbf{Add}(Y, \mathbf{Mult}(X, Y)) \end{aligned}$$

■

Example 2.4 Discrete Interaction Systems are Turing complete. A simple way to prove it is by coding Combinatory Logic. Recall that CL is defined by the two rules: $\mathbf{S}xyz \rightarrow xz(yz)$ and $\mathbf{K}xy \rightarrow x$. These rules do not fit with the rewriting rules of *IS*'s; we must breack them down in more atomic steps, namely:

$$\begin{aligned} @(\mathbf{S}, X) &\rightarrow \mathbf{S}_1(X) & \qquad @(\mathbf{K}, X) &\rightarrow \mathbf{K}_1(X) \\ @(\mathbf{S}_1(X), Y) &\rightarrow \mathbf{S}_2(X, Y) & \qquad @(\mathbf{K}_1(X), Y) &\rightarrow X \\ @(\mathbf{S}_2(X, Y), Z) &\rightarrow @(@(X, Z), @(Y, Z)) \end{aligned}$$

That is, we have five constructors $\mathbf{S}$, $\mathbf{S}_1(-)$, $\mathbf{S}_2(-,-)$, $\mathbf{K}$, $\mathbf{K}_1(-)$ and only one destructor @. Note that the rules above mimic the search for the arguments along the spine of the term. In a sense, they provide the obvious "implementation" of Combinatory Logic. ■

Discrete Interaction Systems have very nice properties. In particular:

1. If there are rewriting rules for all pairs **d-c**, all the *closed* terms in normal form are only composed of constructors. This is particularly relevant in a typed setting, since in this case the constructors of a given type uniquely define the elements of that type. For instance, the natural numbers are defined by the two constructors 0 and $\mathbf{S}(-)$, and the definition of new destructors does not modify the "abstract data type" Nat.

2. It is trivial to implement optimal reductions. Indeed, since we do not have bindings (i.e. the net is acyclic), we never introduce Fan-Out in the reduction. This implies that we do not even need control operators to match Fan-In and Fan-Out.

There is an extremely important point to be understood here. As we remarked above, it is trivial to provide an optimal implementation of Discrete Interaction Systems. Since they are Turing Complete, one may wonder what is the interest to consider higher order systems, where the correct handling of sharing becomes much more difficult. For instance, in the case of λ-calculus, we may compile a λ-term M in a term M' of Combinatory Logic, and then reduce M' in an optimal way. The problem is that the optimal reduction of M' has nothing to do with optimality in the λ-calculus.

Let us consider the following example. We assume the usual encoding of λ-abstraction in CL, namely:

$$\begin{aligned}
[\![\lambda x.\, x]\!] &= \mathbf{SKK} \\
[\![\lambda x.\, M]\!] &= \mathbf{K}M \qquad\qquad \text{if } x \notin \mathbf{var}(M) \\
[\![\lambda x.\, MN]\!] &= \mathbf{S}([\![\lambda x.\, M]\!])([\![\lambda x.\, N]\!])
\end{aligned}$$

Consider a term $M = (\lambda x.\, x(x(y)))(\lambda z.AB)$. If you translate M in CL and you reduce it as a discrete IS according to the rules above, you end up in the situation illustrated in Figure 2. Note now that the

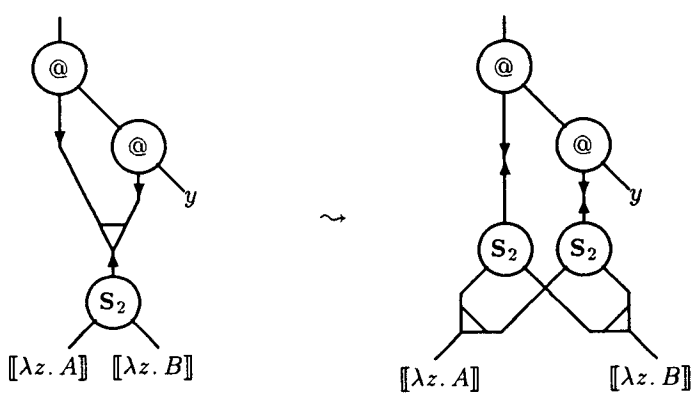


Figure 2:

subterm $S_2([\![\lambda z.\, A]\!], [\![\lambda z.\, B]\!])$ is the translation of $\lambda z.AB$, so the duplication of $S_2(-,-)$ corresponds to a duplication of the application AB. If AB was a redex, this is eventually duplicated. For instance, the optimal reduction of $n(mI)x$ (where n and m are Church integers), once translated in CL, takes a

number of steps which grows as $n * m$, while an optimal implementation of the λ-calculus should take a time proportional to $n + m$.

The previous example depends from the particular encoding of λ-abstraction in CL (but not from our IS-implementation of CL, which is optimal). However, the problem seems to be completely general: when we translate a λ-term M in CL-term M', there will be applications in M which are not translated by applications in M', but coded as arguments of S. When one of these S is shared, it must be duplicated, that implies a blind, possibly unoptimal duplication of the application.

2.5 (Higher Order) Interaction Systems

It has been claimed by Klop that Combinatory Reduction Systems are the higher order generalization Term Rewriting Systems. Then, Interaction Systems are to CRS's as Discrete Interaction Systems are to TRS's.

The main novelties are that every operator of the syntax may now binds variables, and that rewriting rules can be defined by using a meta-operation of substitution, as in the case of the λ-calculus. The formal syntax is defined in the next section, so it is not worth to spend much words, here.

Let us just briefly discuss the main ideas which relate our syntax to the graphical intuition provided by Interaction Nets.

As we have already remarked, the concrete syntax of Interaction Systems is particularly simple, due to their intuitionistic nature. An agent with m input partitions, is represented by a "form" $\mathbf{f}(X_1, \ldots, X_m)$ (the output of the agent is the "root" of the form itself). We have to specify the cardinality of the input partitions, that amounts to specify the number of positive ports in each partition (recall that there is a *single* negative port in each input partition). So each form comes equipped with an "arity" $k_1, \ldots, k_m$. In the discrete case, we have $k_i = 0$ for all i (this forms are called *simple* in [Ac78]).

We have still to specify the principal port for a destructor; without loss of generality we may always assume that it is the first input of the form. This means that, in case of a destructor $\mathbf{d}(X_1, \ldots, X_m)$, we have $k_1 = 0$.

With these provisos, the interaction between constructors and destructors (lhs of the rewriting rules) assume the following typical shape:

$$\mathbf{d}(\mathbf{c}(\vec{x}^1_{k_1}.X_1, \cdots, \vec{x}^m_{k_m}.X_m), \cdots, \vec{x}^n_{k_n}.X_n)$$

where $i \neq j$ implies $X_i \neq X_j$ (*left linearity*). The arity of $\mathbf{d}$ is $0k_{m+1} \cdots k_n$ and that of $\mathbf{c}$ is $k_1 \cdots k_m$.

3 The abstract syntax

As we have already explained in the introduction, Interaction Systems are a subclass of Klop's Combinatory Reduction Systems [Kl80] (see also [Ac78]). In particular, we just add more conditions on the shape of the *left hand sides* of the rewriting rules. So, nothing really exciting is going on at the syntactical level, and we shall rapidly pass through it. The aim of the following exposition is more to

fix the notation, than to provide a completely accurate account of our particular formal system (see [Ac78] and [Kl80] for more details, at an even more abstract level).

An Interaction System is defined by a *signature* Σ and a set of *rewriting rules* R. Both the signature and the rules have to satisfy some constraints which have been suggested by the logical programming discipline which is behind our systems (and, more generally, behind the idea of *interaction* [Laf90]).

(The signature) The signature Σ consists of a denumerable set of *variables* and a set of *forms*. The set of forms is partitioned into two disjoint sets Σ^+ and Σ^-, representing *constructors* (ranged over by $\mathbf{c}$) and *destructors* (ranged over by $\mathbf{d}$), respectively. Variables will be ranged over by x, y, z, $\cdots$, possibly indexed. Vectors of variables will be denoted by $\vec{x}_i^j$ where j is used to discriminate between vectors and i states the lenght of the vector. Thus we assume that $\langle x_1^j, \cdots, x_i^j \rangle$ be the vector corresponding to $\vec{x}_i^j$. We will omit the index i when it can be derived from the context or it is not essential for what goes on. Similarly we will not write empty vectors.

Each form can work as a binder. This means that in the arity of the form we must specify not only the number of arguments, but also, for each argument, the number of variables it is supposed to bind. Thus, the *arity* of a form $\mathbf{f}$, is a finite (possibly empty) sequence of naturals (and not, as usual, a natural!). Moreover, we have the constraint that the arity of every destructor $\mathbf{d} \in \Sigma^-$ has a leading 0 (i.e., it cannot bind over its first argument). The reason for this restriction is that, in Lafont's notation [Laf90], at the first argument we find the *principal port* of the destructor, that is the (unique) port where we will have interaction.

Expressions, ranged over by $t, t_1, \cdots$, are inductively generated by the two rules below:

a. every variable is an expression;

b. if $\mathbf{f}$ is a form of arity $k_1 \cdots k_n$ and $t_1, \cdots, t_n$ are expressions then $\mathbf{f}(\vec{x}_{k_1}^1.t_1, \cdots, \vec{x}_{k_n}^n.t_n)$ is an expression.

Thus the arity $k_1 \cdots k_n$ of a form gives, from one side, the number of arguments it takes (the length n of the string) and, on the other, the "amount of bindings" k_i it is able to do over each argument t_i.

Free and bound occurrences of variables are defined in the obvious way. Namely, let $\mathbf{var}$ be the function from expressions to sets of variables defined inductively as follows:

$$\begin{aligned} \mathbf{var}(x) &= \{x\} \\ \mathbf{var}(\mathbf{f}(\vec{x}^1.t_1, \cdots, \vec{x}^n.t_n)) &= \bigcup_i (\mathbf{var}(t_i) \setminus \vec{x}^i) \end{aligned}$$

We will say that x is *free* in t if $x \in \mathbf{var}(t)$ and x is bound in the i-th argument of $\mathbf{f}(\vec{x}^1.t_1, \cdots, \vec{x}^n.t_n)$ when $x \in \vec{x}^i$ (the notation for "x is an element of $\vec{x}^{i}$").

Terms which differs for the name of bounded variables will be identified. That is, we assume that *IS*'s are quotiented by the axiom of α-conversion below

$$\mathbf{f}(\cdots, \vec{x}_n.t, \cdots) = \mathbf{f}(\cdots, \vec{y}_n.t[{}^{y_1}/_{x_1}, \cdots, {}^{y_n}/_{x_n}], \cdots) \qquad \vec{y}_n \cap \mathbf{var}(t) = \varnothing$$

Observe that, as in the case of the λ-calculus, it is assumed the presence of a metaoperation of *substitution* which allows us to replace terms for variables. Substituition plays a central role in the

following rewriting rules, which are to be considered as a generalization of the β-conversion of the λ-calculus. As usual, the metaoperation of substitution $t[^{\vec{N}}/_{\vec{x}}]$ is defined inductively over the syntax of t. Let $[^{\vec{N}}/_{\vec{x}}]$ denote the vector of substitutions $[^{N_1}/_{x_1}, \cdots, {}^{N_n}/_{x_n}]$ (the variables x_i are taken pairwise distinct): the simple case is when t is a variable. Then

$$\begin{array}{rcll} x[^{\vec{N}}/_{\vec{x}}] & = & N_i & x = x_i \\ y[^{\vec{N}}/_{\vec{x}}] & = & y & y \notin \vec{x} \end{array}$$

When t is not a variable, we have to push the substitution inside the outermost form. Here is the axiom:

$$\mathbf{f}(\vec{x}^1_{k_1}.t_1, \cdots, \vec{x}^n_{k_n}.t_n)[^{\vec{N}}/_{\vec{x}}] = \mathbf{f}(\vec{y}^1_{k_1}.t'_1, \cdots, \vec{y}^n_{k_n}.t'_n)$$

where

1. $\vec{y}^i_{k_i} = \vec{x}^i_{k_i}$ and $t'_i = t_i$ whenever $\vec{x} \subseteq \vec{x}^i_{k_i}$;
2. $\vec{y}^i_{k_i} = \vec{x}^i_{k_i}$ and $t'_i = t_i[^{\vec{N}}/_{\vec{x}}]$ whenever $\vec{x} \cap \vec{x}^i_{k_i} = \varnothing$ and ($\vec{x} \cap \mathbf{var}(t_i) = \varnothing$ or $\mathbf{var}(N) \cap \vec{x}^i_{k_i} = \varnothing$);
3. $t'_i = t_i[^{\vec{y}^i_{k_i}}/_{\vec{x}^i_{k_i}}][^{\vec{N}}/_{\vec{x}}]$ whenever $\vec{x} \cap \vec{x}^i_{k_i} \neq \varnothing$ and $\vec{x} \cap \mathbf{var}(t_i) \neq \varnothing$ and $\mathbf{var}(N) \cap \vec{x}^i_{k_i} \neq \varnothing$) and $\vec{y}^i_{k_i} \cap (\mathbf{var}(t) \cup \mathbf{var}(N)) = \varnothing$.

(above $\mathbf{var}(N) = \bigcup_i \mathbf{var}(N_i)$).

(The rewriting rules) Rewriting rules are described by using schemas or *metaexpressions*. A metaexpression is an expression built up with *metavariables*. Metavariables will be ranged over by X, Y, $\cdots$, possibly indexed. From a syntactical point of view, metavariables may be considered as forms of arity ϵ. A more correct approach would consist in considering metavariables together with their specified bounded variables, i.e. structures of the kind $\langle x_1, \ldots, x_n \rangle.X$ or also $X[\,/_{x_1}, \ldots, {}_{x_n}]$, according to the point of view. In this case, this "metavariables" should be considered as forms of arity 0^n (the string of n zero's); see [Ac78] for more details.

A *rewriting rule* is a pair of metaexpressions, written $H_1 \rightarrow H_2$, where H_1 (the *left hand side* of the rule) has the following format

$$\mathbf{d}(\mathbf{c}(\vec{x}^1_{k_1}.X_1, \cdots, \vec{x}^m_{k_m}.X_m), \cdots, \vec{x}^n_{k_n}.X_n)$$

where $i \neq j$ implies $X_i \neq X_j$ (*left linearity*), the arity of X_i is k_i whilst the arity of $\mathbf{d}$ is $0k_{m+1} \cdots k_n$ and that of $\mathbf{c}$ is $k_1 \cdots k_m$. The *right hand side* H_2 is every metaexpression without free variables (or *closed*, according to Klop's terminology, [Kl80]) built up by the following syntax:

$$H \;::=\; x \;\mid\; \mathbf{f}(\vec{x}^1_{a_1}.H_1, \cdots, \vec{x}^j_{a_j}.H_j) \;\mid\; X_i[^{H_1}/_{x^i_1}, \cdots, {}^{H_{a_i}}/_{x^i_{a_i}}]$$

where x is a generic variable, $\mathbf{f}$ is a form of arity $a_1 \cdots a_j$ and $1 \leq i \leq n$. Notice that, in the case of λ-calculus only the third alternative is exploited. The generalization here mainly concerns *creation of new forms* by rules, i.e. a new form $\mathbf{f}$ which does not appear in the initial expression comes out as a consequence of the rewriting rule. We will say that $\mathbf{f}$ is *created* by the rule. This phenomenon does not appear in λ-calculus where β-reduction *at most* duplicates *old* forms (i.e. already existing in the lhs).

Notice that, according to the above syntax, the metavariables which appear in the rhs are a subset of those appearing in the lhs.

We will also assume that H_2 is in normal form with respect to substitutions (i.e. substitutions are pushed inside the metaexpresions as much as possible). Of course this restriction by no means affects the generality of our systems.

Finally, the sets R of rewriting rules will be taken as ***non-ambigous***, i.e. there exists at most one rewriting rule for every pair **d** - **c**.

Example 3.1 (The λ-calculus) Perhaps the most important example of an *IS* is the λ-calculus. In this case we have two forms: @ and λ. The first has arity 00, the latter has arity 1. There is just one rewriting rule: the β-reduction whose format is:

$$@(\lambda(\langle x\rangle.X(x)),\, Y) \;\rightarrow\; X[^{Y}/_{x}].$$ ■

Example 3.2 Interesting Interaction Systems can be defined by enriching the λ-calculus. For instance, it is possible to enrich the λ-calculus with an *if-then-else* operator $\natural$. Let **T** and **F** be two (new) constructors of arity ε and $\natural$ be a destructor of arity 000. The rules modelling the conditional behaviour are due to the interactions between $\natural$ and **T** or **F** as shown below:

$$\natural(\mathbf{T},\, X,\, Y) \rightarrow X$$

$$\natural(\mathbf{F},\, X,\, Y) \rightarrow Y$$

Note that we do not have any interaction between $\natural$ and λ.

A less trivial example is provided by the recursion operator μ. We may describe it as a destructor of arity 0, interacting with λ as follows:

$$\mu(\lambda(\langle x\rangle.X)) \rightarrow X[^{\mu(\lambda(\langle x\rangle.X)}/_{x}]$$

Note that the two forms λ and μ in the rhs have nothing to do with the homonymous forms in the lhs. Actually, a more correct writing of the previous rule would be the following:

$$\mu(\lambda(\langle x\rangle.X)) \rightarrow X[^{\mu(\lambda(\langle y\rangle.X[^{y}/_{x}])}/_{x}]$$

The previous encoding of μ in IS's is not the only solution, and not even the best one (see Section 11), but we found instructive and amusing to describe it as another destructor for λ, apart from application.

An alternative way to treat recursion in IS's, is with the Y operator. In this case, Y is considered as a constructor of arity ε interacting with @ in the following way:

$$@(Y, M) \rightarrow @(M, @(Y, M)).$$ ■

3.1 The graphical representation

As we discussed in section 1, the following graphical representation of terms derives from the logical paradigm underlying Interaction Systems, and in particular from their relations with Interaction Nets.

A form **f** of arity $k_1 \ldots k_n$ is represented as a node of name **f** with $1 + \sum_{i=1}^{n} p_i$ ports (edges); $p_i = k_i + 1$ is the i-th *partition* of **f**. The only port which does not belong to a partition is drawn with an outgoing arrow, and it is called the *principal port* of the form **f**; the other entries are called *auxiliary ports* (see [Laf90]). The i-th partition represents the connections between **f** and its i-th argument M. In particular, one connection is with the root of M, and k_i with the variables bound by **f**. From the graphical point of view we have several possibilities for representing the connection between bound variables and their binders. Let us briefly discuss them.

1. The connection is left implict. This is the usual approach based on abstract syntax trees. The edge exiting from the binder leads to a "formal" variable with a specified name, and this name is used to establish the actual "connection" with the variables inside the argument. This approach creates problems with α-conversion, during the reduction.

2. The connection is explicitly drawn. One drawback of this approach is that the number of links at a bounded port of a given form is arbitrary, and it may change during the computation.

3. In order to solve the previous problem, we could add "sharing nodes" in the graph taking care of the multiple occurrences of a same variable. This is the solution we shall adopt in the implementation, and the one that provides the correct intuition. However, this approach implies an explicit handling of this new operators, that is not worth discussing, for the moment.

We shall adopt the second solution. So bound variables correspond to ports of the binding form **f**. This means that our graphs are cyclic. Remark again that the number of links at a bounded port of a form is arbitrary and variable.

If **f** is a constructor, the principal port is "at the root" of the form; if **f** is a destructor, its principal port is at its first argument (recall that no binding is possible over this argument). So, interaction between constructors and destructors takes place only at principal ports (local sequentiality).

The correspondence between ports, bound variables and body of the arguments is fixed once and for all for each form. This means that all ports of a given form should be suitably "marked" (for the sake of readibility, we shall generally omit to do that).

Following [Laf90], it is possible to attribute a "sign" + or − to every port of a form. In particular, the principal port of a constructor **c** is always positive (the em output of **c**), while the principal port of a destructor is always negative. Every partition contains *at most* one negative port. If it contains (exactly) one negative port, it is called an *input*-partition, and an *output*-partition otherwise. Output partitions are always singletons.

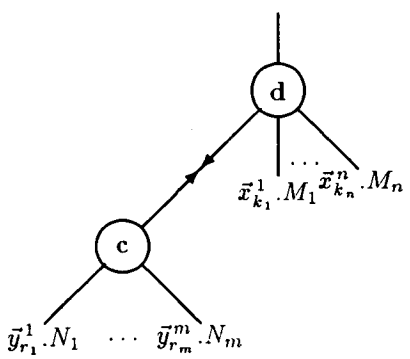
All the partitions of a constructor **c** are inputs (since it already has its "output" at the principal port). Every destructor **d** has exactly one output (a singleton partition), at its "root".

These constraints are naturally imposed by the fact of considering *intuitionistic* systems. In particular, every form must have a *unique* output (functionality). In an input-partition of arity $p_i = k_i + 1$ of some form **f**, the unique negative port leads to the output (the "root") of its i-th argument M. The remaining k_i positive ports of the input-partition leads to the k_i distinct variables bounded by **f** in M.

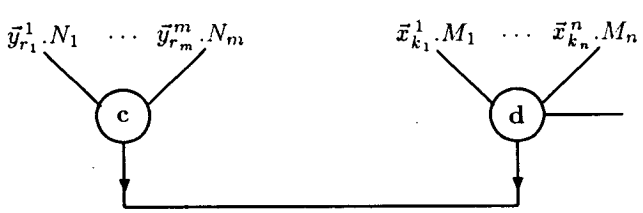
The important fact is that every graphical connection between forms is always through ports of opposite sign, as the reader can easily verify by himself.

Partitions and signs are relevant for the development of the theory, and in particular for several aspects of static semantics (consistency, typing, absence of deadlocks, and so on). On the contrary, they are not essential from the point of view of reduction. For this reason, we shall often omit to specify them in our graphical representation (unless it could be convenient for the sake of clarity).

There is still one point worth to be discussed, about the *orientation* of forms. Indeed, when considering intuitionistic systems such as IS's, there are at least two natural approaches. The first one, essentially inspired by the usual representation of "abstract syntax trees", consists to put the (unique) output up, and all inputs(-partitions) down (and we have already informally relied on this approach, identifying "outputs" and "roots" of subterms). In this case, a typical interaction between constructors and destructors in IS's is represented as follows (the following picture is an over-simplification of the graphical representation discussed above; in particular, each partition has been represented by a single edge).



The previous representation is natural, and almost standard in Computer Science. The main drawback, is that, being directly inspired by the syntax, it hides the *essential symmetry* between constructors and destructors in Interaction Systems. The alternative approach is to follow the usual *Proof Nets* notation, where principal ports are oriented down, and auxiliary ports are oriented up. In this case, we would obtain the following *symmetric* picture for a typical redex **d-c**:



The proof net notation is particularly elegant, but it often results surprisingly ackward for people not really acquainted with it. Actually, one of the reason of the initial, chilly reception of Proof Nets from the Computer Science Community was due to an essential misunderstanding of the previous notational modification (see [As91] for a discussion). In the following, we shall indifferently use both notations.

3.2 Residuals and Standardization

Given an *IS* (Σ, R) and an expression t in Σ, we denote subexpressions of t by their access path in the syntactic tree of t. Access paths will be ranged over by u, v, w, etc. We write that $u \prec v$ if the string u is a *prefix* of the string v. The notation $u|v$ stays for $u \not\prec v$ and $v \not\prec u$.

A subexpression at u in t is a *redex* if it matches with the lhs of a rewriting rule r in R, i.e. if the lhs of r has the form $\mathbf{d}(\mathbf{c}(\vec{x}^1_{k_1}.X_1, \cdots, \vec{x}^m_{k_m}.X_m), \cdots, \vec{x}^n_{k_n}.X_n)$ then the subexpression at u of t is $\mathbf{d}(\mathbf{c}(\vec{y}^1_{k_1}.t_1, \cdots, \vec{y}^m_{k_m}.t_m), \cdots, \vec{y}^n_{k_n}.t_n)$ (for some $\vec{y}^i_{k_i}$ and t_i) and we say that $\vec{x}^i_{k_i}$ and X_i are instantiated by $\vec{y}^i_{k_i}$ and t_i, respectively. The requirement that the set R is unambiguous implies the existence of at most one rule matching a given subexpression. In the above case, the *reduction* at u, that is the firing of the redex at u, consists of replacing the subexpression at u in t by the instance of the rhs of r, i.e. the expression yielded by replacing every occurrence of (variables in) $\vec{x}^i_{k_i}$ and X_i in the rhs of r by (the corresponding variable in) $\vec{y}^i_{k_i}$ and t_i, respectively. Reductions will be denoted by boldface letters of the corresponding redexes: as much as possible we will not care of initial and final terms of reductions, assuming that they are always well-defined.

The basic notion for what follows is the definition of *residual*. Roughly, residuals formalize what remains of a redex after a reduction. In view of this definition, let us define the set $o_x(t)$ of occurrences of the free variable x inside an expression t by structural induction over t, as follows

$$o_x(y) = \begin{cases} \{\varepsilon\} & \text{if } x = y \\ \varnothing & \text{otherwise} \end{cases}$$

$$o_x(\mathbf{f}(\vec{x}^1_{k_1}.t_1, \cdots, \vec{x}^n_{k_n}.t_n)) = \bigcup_{\substack{1 \le i \le n \\ x \notin \vec{x}^i_{k_i}}} i\,o_x(t_i)$$

where concatenation of a natural i with a set S of strings is the set $\{is \mid s \in S\}$ and $i\varnothing$ is, by definition, equal to $\varnothing$.

Definition 3.3 Let t be an expression of an *IS* (Σ, R) and u and v be two redexes in t. The set $v/\mathbf{u}$ of *residuals* after the reduction $\mathbf{u}$ (whose schema is r) is a set of redex occurrences inside the ending expression of $\mathbf{u}$ defined by cases as follows:

$$v/\mathbf{u} = \begin{cases} \varnothing & \text{if } u = v \\ \{v\} & \text{if } u|v \text{ or } v \prec u \\ \mathcal{O}^t_{u,v}(H) & \text{if } v = uw \text{ and } H \text{ is the rhs of } r \end{cases}$$

where the set $\mathcal{O}^t_{u,v}(H)$ is defined by induction over the structure of H by the following rules (notice that v has to occur inside the instance of a metavariable of r):

1. $\mathcal{O}^t_{u,v}(H) = \varnothing$ when $H = x$, for every variable x;
2. $\mathcal{O}^t_{u,v}(H) = \bigcup_i i\mathcal{O}^t_{u,v}(H_i)$ when $H = \mathbf{f}(\vec{x}^1.H_1, \cdots, \vec{x}^n.H_n)$;

3. $\mathcal{O}^t_{u,v}(H) = S \cup (\bigcup_i \{w_i \mathcal{O}^t_{u,v}(H_i) \mid w_i \in o_{x_i}(t_j)\}$, when $H = X_j[^{H_1}/_{x_1}, \cdots, ^{H_n}/_{x_n}]$ and where $S = \varnothing$ if v does not occur inside t_j (the subterm of t which replaces X_j in the instance of r) or $S = \{v''\}$ if $v = uv'v''$ and v' is the access path of the variable X_j in the rhs of r. ■

The operation of residual can be smoothly generalized to sets:

$$U/\mathbf{u} = \bigcup_{v \in U} v/\mathbf{u}.$$

Furthermore, for every derivation σ, i.e. a sequence of 0, one or more reductions, the set of residuals of u after σ (notation u/σ) is defined inductively by:

1. $u/\varepsilon = \{u\}$;
2. $u/(\sigma\,;\mathbf{v}) = \{w \mid w \in v'/\mathbf{v} \ \wedge \ v' \in u/\sigma\}$.

(ε is the empty derivation). Let U be a set of redexes in t_1. A derivation $\sigma = \mathbf{u}_1\,;\mathbf{u}_2\,;\ldots;\mathbf{u}_n\,;\cdots$ is *relative* to U iff for every $n \geq 1$, $u_n \in U/(\mathbf{u}_1\,;\mathbf{u}_2\,;\ldots;\mathbf{u}_{n-1})$. The sequence σ is a ***development*** of U iff σ is relative to U and $U/\sigma = \varnothing$.

Notation We will write $t \xrightarrow{u} t'$ when we want to emphasize the initial and ending expressions of a reduction $\mathbf{u}$. Similarly, the notation concerning derivations σ will be $t \xrightarrow{\sigma} t'$.

Below, we briefly recall some results about orthogonal (i.e. left linear and without critical pairs) *Combinatory Reduction Systems* mainly due to Klop in [Kl80]. The main purpose of CRS's is to study, in a general framework, the problems of α-conversion and substitution. CRS's generalize IS's by relaxing our constraints over the format of rules.

Theorem 3.4 [Kl80] Every development in a *CRS* is finite and end at the same term. Moreover, if σ and ρ are two developments then, for every redex u, $u/\sigma = u/\rho$. ■

Therefore the order according to which redexes are contracted is not relevant both for the effect over the final state and over other redexes. This means that we can unambiguously define the notion of parallel reduction contracting in one step a set U of redexes (notation $\mathbf{U}$). In particular, the final expression of $\mathbf{U}$ is that reached by an arbitrary development of U and, similarly, when σ is a development of U, $u/\mathbf{U}$ is equal to u/σ.

The following lemma states a diamond property between parallel moves.

Lemma 3.5 Let U and V be two sets of redexes in the same term t. Then

1. $\mathbf{U};(\mathbf{V}/\mathbf{U})$ and $\mathbf{V};(\mathbf{U}/\mathbf{V})$ end at the same term;
2. for every set W of redexes in t, $W/(\mathbf{U};(\mathbf{V}/\mathbf{U})) = W/(\mathbf{V};(\mathbf{U}/\mathbf{V}))$.

Proof: Straightforward application of Theorem 3.4 because both $\mathbf{U};(\mathbf{V}/\mathbf{U})$ and $\mathbf{V};(\mathbf{U}/\mathbf{V})$ are two developments of the set $U \cup V$. ■

We denote with $\mathbf{U} \sqcup \mathbf{V}$ the derivation $\mathbf{U};(\mathbf{V}/\mathbf{U})$. The residual operation can be generalized over pairs of coinitial derivations, too. The definition is given inductively over the sum of lenghts of σ and ρ by

$$\sigma/\rho = \begin{cases} \varepsilon & \text{if } \sigma = \varepsilon \\ (\sigma'/\rho);(u/(\rho/\sigma')) & \text{if } \sigma = \sigma';\mathbf{u}. \end{cases}$$

Here is the definition of permutation equivalence.

Definition 3.6 The *permutation equivalence* (notation $\equiv$ is the least equivalence relation satisfying:

1. $\mathbf{U} \sqcup \mathbf{V} \equiv \mathbf{V} \sqcup \mathbf{U}$;
2. $\varepsilon \equiv \varnothing$;
3. if $\sigma \equiv \rho$ then $\tau;\sigma;\tau' \equiv \tau;\rho;\tau'$. ■

CRS's enjoy a strong form of the Church-Rosser theorem, stating the existence of permutation equivalent diamonds. Theorem 3.9 below guarantees that a "standard" reduction can always be found for every equivalence class. This latter theorem holds for *left normal CRS* only, i.e. those *CRS* whose lhs of rewriting rules have no form as argument of a metavariable (forms are all "to the left"). Of course, *IS* are a subclass of left normal *CRS*. In Section 5 we will found another example of left normal *CRS*'s: the labelled *IS*'s.

Theorem 3.7 (Strong Church-Rosser) For every pair σ and ρ of coinitial derivations, $\sigma;\rho/\sigma \equiv \rho;\sigma/\rho$ holds.

Proof: Iterating Lemma 3.5 it is possible to prove that both $\sigma;\rho/\sigma$ and $\rho;\sigma/\rho$ end at the same term. By definition of permutation equivalence over these iterations, we yield $\sigma;\rho/\sigma \equiv \rho;\sigma/\rho$. ■

Before stating the Standardization Theorem for left normal *CRS*, we need of a preliminar lemma. To this aim, given a derivation $\sigma = \mathbf{v}_0;\mathbf{v}_1;\cdots$ having t as initial expression, then a redex u in t is *contracted* by σ if, for some i, $v_{i+1} \in u/(\mathbf{v}_0;\cdots;\mathbf{v}_i)$. Moreover, u is *consumed* by σ if $u/\sigma = \varnothing$. Then, we define $\mathbf{lmc}(\sigma)$ as the leftmost-outermost redex in t contracted by σ.

Lemma 3.8 [Kl80] Let σ be a derivation in a left normal *CRS* having t as initial expression and let $u = \mathbf{lmc}(\sigma)$. Then

1. for every σ' such that $\sigma = \sigma';\sigma''$, $u/\sigma' = \varnothing$ and u is contracted in σ' or $u/\sigma' = \mathbf{lmc}(\sigma'')$;
2. u is consumed in σ. ■

Theorem 3.9 Standardization Theorem [Kl80] Every derivation σ in a left normal *CRS* is permutation equivalent to the unique one firing always the leftmost-outermost redex fired by σ.

Proof: Take the derivation σ_S defined through the following procedure:

$$\sigma_0 = \sigma \qquad v_0 = \mathbf{lmc}(\sigma)$$
$$\sigma_{i+1} = \sigma_i/v_i \qquad v_{i+1} = \mathbf{lmc}(\sigma_{i+1})$$

Thus σ_S of σ is $\mathbf{v}_0; \mathbf{v}_1; \cdots$. Lemma 3.8 guarantees that σ_S is well-defined: in particular we yield that, at each step, the leftmost-outermost redex can not be consumed without being contracted and that it is consumed in exactly one contraction. Hence, the following points remain to be proved:

1. σ_S is finite and always reduces the leftmost-outermost redex fired by σ;
2. $\sigma_S \equiv \sigma$;
3. σ_S is unique.

The proof of 1, 2 and 3 is similar to the case of λ-calculus. ■

Given a derivation σ, we will call *standard* the derivation obtained by the algorithm in the proof of Theorem 3.9.

4 Redex families: an introduction

The notion of family tries to capture the phenomenom of *duplication* of redexes typical of all non-linear rewriting systems. In a sense, two redexes are in a same family if and only if, they are "copies" of a same redex. Although this notion is quite intuitive, it is not easily formalizable. We start this section with a remind of several possible definitions of "family" which have been proposed in the literature. Form many systems, such as the λ-calculus and orthogonal term rewriting systems, these notions turn out to be equivalent. Surprisingly, this is not the case of IS (see next section).

(zig-zag) This is the most abstract definition of the notion of family [Le78]. Duplication of redexes is formalized as residuals modulo permutations. In particular, a redex u with history σ (notation σu) is a *copy* of a redex v with history ρ iff $\rho v \leq \sigma u$. The family relation is then the symmetric and transitive closure of the copy-relation (pictorially, this gives rise to the "zig-zag"). Note that, intuitively, the family relation has to be an equivalence relation.

(extraction) Here the intuition is that two redexes should be in the same family if and only if they have been created "in the same way", that is if the "relevant" part of their histories coincide [Le80]. This is formalized by defining an "extraction" relation over redexes (with history) σu, which trows away all the redexes in σ which have not been relevant for the creation of u. One problem with this definition is that it only works when standard derivations are considered. Note that a canonical form for σu w.r.t the extraction relation essentially express the causal dependencies of u along the derivation. The fact of considering standard derivations, allows us to deal with *sequences* instead of *partial orders*.

(labelling) A suitable labelling of subexpressions is introduced, together with a labelled variant of the calculus under investigation [Le78, Kl80]. The idea of labels is essentially that of marking the "access path" to a given information. Labels grow along the reduction, keeping a trace of what has been done in order to get a subexpression in a given position. Then, two redexes are in a same family if the "access paths" to the subexpressions involved in the reduction are the same. This is particularly clear

in Interaction Systems, where the relevant path of a redex is that leading from the destructor to the constructor.

(sharing) This is the most operational notion, since it amounts to define an "optimal" evaluator for the calculus. Two redexes are in a same family w.r.t this approach if their reduction is shared by the evaluator. Optimal evaluators are known since a very long time for recursive program schemes [Vu74] and Combinatory Logic [St74], but it is only in recent time that these results have been extended to the λ-calculus [Lam90, Ka90, GAL92]. The main aim of our research is the definition of an optimal evaluator for Interaction Systems, providing in this way an optimal implementation for a formal language comprising all the previous ones.

4.1 Problems with Interaction Systems

Surprisingly, the theoretical characterization of the family relation in Interaction Systems is much more complicated of what one could imagine. The only approach which has a (more or less) smooth generalization to IS's is that based on labelling. All the other notions create problems. Let us briefly consider them.

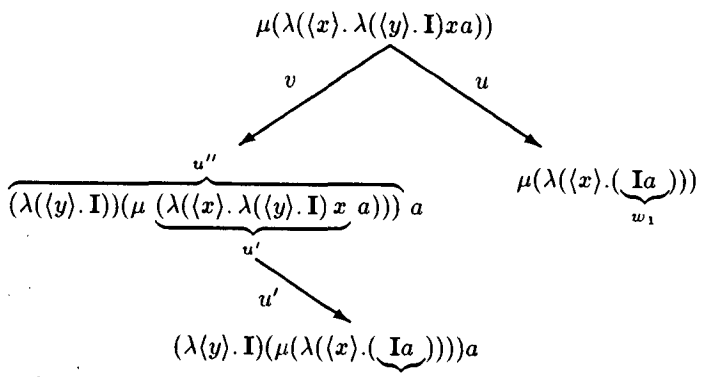
The zig-zag is simply inadequate. This proves that, inspite of its abstract nature, it is not general enough. We have been so far unable to provide an elegant (and it must be *elegant*!) generalization of the zig-zag relation to *IS*.

Here is a counterexemple.

Example 4.1 (The operator μ). We consider again the λ-calculus extended with a recursion operator μ as defined in Example 3.1 (for simplicity, the shall write the application in the usual infix form). Take the following term:

$$\underbrace{\mu(\lambda(\langle x\rangle.\overbrace{(\lambda(\langle y\rangle.\mathbf{I})x}^{u}\, z)))}_{v}$$

where $\mathbf{I} = \lambda(\langle x\rangle.\, x)$ and the two redexes are explicitely named by the under/over-braces. Now, by using



Lévy's notion of family, there is no way to relate the two redexes w_1 and w_2 (which are *intuitively* in the same family). Indeed if we attempt to complete the derivation $\mathbf{v};\mathbf{u}'$ by firing the redex u'', the

redex w_2 is cancelled. The interesting fact is that this phenomenon does not happen with the operator $\mathbf{Y}$ and the rule $\mathbf{Y}f \longrightarrow f(\mathbf{Y}f)$. This because now the above reduction v is splitted in simpler moves and the new intermediate states allows us to make the right 'zig-zag'. Following this idea, you may derive plenty of counterexamples to the zig-zag notion of family in Interaction Systems. It is enough to add to the λ-calculus new "combinators" performing several steps of β-reduction at a time. ■

Let us consider another counterexample, showing a different phenomenon.

Example 4.2 Take the term $M = \mu(\lambda(\langle x\rangle.\, xx))$. Then, by firing the unique redex u in M, we obtain

$$M' = (\mu(\lambda(\langle x\rangle.\, xx)))(\mu(\lambda(\langle x\rangle.\, xx)))$$

where the two redexes v' and v'' should intutively belong to a same family. Indeed, the rewriting rule concerning the pair μ-λ creates a redex (a new pair μ-λ) in the rhs which has to be substituted for the bound variable x. Of course, according to the number of occurrences of x in the body of the λ, we yield copies of a *same* redex. Again, the notion of zig-zag does not cope with this situation. ■

Example 4.2 also shows some of the problems we find in generalizing Lévy's *extraction process.* Indeed the redex u is essential for the creation of the two redexes v' and v'' in M'. This means that the two histories uv' and uv'' are already in *canonical form* in Lévy's sense, and this canonical forms are unfortunately different.

For a time, we hoped to solve this problem by adding a new operation of normalization over histories. The idea was to "shift" a reduction internal to some instance of a shared subterm (an arguments of some substitution) in the rhs of a rewriting rule to its isomorphic reduction in the leftmost outermost instance of the same subterm. Unfortunately, this generalization of the extraction process is not general enough, as it is proved by the following counterexample.

Example 4.3 Consider the λ-calculus extended with the following rewriting rules:

$$d(c(Z, \langle x\rangle.X), \langle y\rangle.Y) \rightarrow d_1(Z, \langle z\rangle.X[^{Y[^z/_y]}/_x])$$

$$d_1(c_1(Z), \langle x\rangle.X) \rightarrow X[^Z/_x]$$

Consider the initial term

$$M = d(c(c_1(\lambda(\langle x\rangle.x)), \langle x\rangle.@(x,x)), \langle y\rangle.@(y,y))$$

After two reductions, we get the term

$$M' = @(@(\lambda(\langle x\rangle.x), \lambda(\langle x\rangle.x)), @(\lambda(\langle x\rangle.x), \lambda(\langle x\rangle.x)))$$

The two redexes in M' are *intuitively* in a same family. Indeed M' should be read as

$$M' = @(x,x)[^{@(z,z)}/_x][^{\lambda(\langle x\rangle.x)}/_z]$$

We leave to the reader to check that the shift operation, as defined above, is not powerful enough to cover this case. ■

The problem in example 4.3 is essentially due to the possibility, provided by IS's, of substituting over metavariables inside new forms in the rhs of the rewriting rules. We have actually proved that, with this restriction, the shift operation works well (but we shall not address the proof in this paper). However, we do not have, at present, a generalization of the extraction process to arbitrary Interaction Systems.

The only approach to the family relation that is easily generalizable from λ-calculus to IS's is labelling. This will be the subject of the next section.

5 Labelled interaction systems

We shall devote this section to the generalization of Lévy's labelling (and some interesting results concerning them) to arbitrary Interaction Systems.

Definition 5.1 Let $L = \{a, b, \cdots\}$ be a denumerable set of *atomic labels*. The set $\mathbf{L}$ of *labels*, ranged over by ℓ, $\ell_1, \cdots$, is defined by the following rules:

$$L \quad | \quad \ell_1\ell_2 \quad | \quad (\ell)_s$$

where $s \in I\!N^+$.

The operation of concatenation $\ell_1\ell_2$ will be assumed to be associative. The set $\mathbf{L}_p$ of *proper labels* contains exactly all those labels $\ell \in \mathbf{L}$ such that ℓ is atomic or $\ell = (\ell')_s$ for some ℓ' and s. The set $\mathbf{L}_p$ will be ranged over by $\alpha, \beta, \cdots$. ■

For istance $(a(bab)_3)_{12}$ is a label. Parentheses play the same role of underlining and overlining in Lévy's labels (see example below).

Although its formalization is a bit entangled, the idea behind the following labelling is very simple. When a redex is fired, a label ℓ is captured between the destructor and the constructor; this is the label associated with the redex. Then, the lhs of the rewriting rule must be suitably "marked" with ℓ, in order to keep a trace of the history of the creation. Moreover, since in the lhs we may introduce new forms, we must guarantee a property similar to the initial labelling, where all labels are different. This means that all links in the lhs must be marked with a different function of ℓ (and we shall use sequences of integers, for this purpose).

Let us come to the formal definition. Every *IS* (Σ, R) can be turned in a free way into a *(labelled) CRS*. (Σ^L, R^L). (Σ^L, R^L) fails to be a IS for quite trivial reasons: we shall define in appendix A a very simple simulation of (Σ^L, R^L) by means of an Interaction System.

The forms of Σ^L are those in $\Sigma \cup \mathbf{L}_p$ with the arity of every $\alpha \in \mathbf{L}_p$ equal to 0. If

$$\mathbf{d}(\mathbf{c}(\vec{x}^1.X_1, \cdots, \vec{x}^m.X_m), \cdots, \vec{x}^n.X_n) \ \rightarrow \ H$$

is a rule in R then, for every i and for every i-tuple $\alpha_1, \cdots \alpha_i$, the rule

$$\mathbf{d}(\alpha_1(\cdots(\alpha_i(\mathbf{c}(\vec{x}^1.X_1, \cdots, \vec{x}^m.X_m)\cdots), \cdots, \vec{x}^n.X_n) \ \rightarrow \ \mathcal{L}^0_{\alpha_1\cdots\alpha_i}(H)$$

belongs to R^L, where $\mathcal{L}^s_\alpha$ is defined over metaexpressions as follows

$$\begin{aligned}
\mathcal{L}^s_\alpha(x) &= (\alpha)_s(x) \\
\mathcal{L}^s_\alpha(\mathbf{f}(\vec{x}^0.H_0,\cdots,\vec{x}^m.H_m)) &= (\alpha)_s(\mathbf{f}(\vec{x}^0.\mathcal{L}^{s0}_\alpha(H_0),\cdots,\vec{x}^m.\mathcal{L}^{sm}_\alpha(H_m)) \\
\mathcal{L}^s_\alpha(X[{}^{H_0}/_{x_0},\cdots,{}^{H_n}/_{x_n}]) &= (\alpha)_s(X[{}^{\mathcal{L}^{s0}_\alpha(H_0)}/_{x_0},\cdots,{}^{\mathcal{L}^{sn}_\alpha(H_n)}/_{x_n}])
\end{aligned}$$

Example 5.2 Consider again the λ-calculus. The β-reduction $@(\lambda(\langle x\rangle.X),Y) \to X[{}^{Y}/_{x}]$ gives rise, in the labelled version, to the following rules:

$$@(\alpha_1(\cdots(\alpha_i(\lambda(\langle x\rangle.X)\cdots),Y) \to \mathcal{L}^0_\ell(X[{}^{Y}/_{x}])$$

where $\ell = \alpha_1\cdots\alpha_i$. Note that, by definition, $\mathcal{L}^0_\ell(X[{}^{Y}/_{x}]) = (\ell)_0(X[{}^{(\ell)_{00}(Y)}/_{x}])$, thus, by replacing ℓ_0 with $\overline{\ell}$ and ℓ_{00} with $\underline{\ell}$, we easily recognize Lévy's labelling. ■

Our handling of sequences of labels may look a bit tricky, but it avoid to introduce the annoying metaoperation of concatenation, with the usual problems created by associativity.

Remark 5.3 Klop has remarked that in the case of the λ-calculus a single parenthesis for labels was enough [Kl80]. In general, this is not the case for Interaction Systems, where we are forced to provide different labels for all new links generated by a rewriting rule (for all new subexpressions). ■

It is trivial to verify that (Σ^L, R^L) is a *left normal CRS*: the corollary below is thus an immediate consequence of Theorem 6.2.8.9 in [Kl80].

Corollary 5.4 In every (Σ^L, R^L) leftmost-outermost reduction is standard. ■

Given a form $\mathbf{f}$ in Σ and an occurrence u of it in a term t of Σ^L, we say that $\mathbf{f}$ has *label* $\alpha_1\alpha_2\cdots\alpha_i$ if $u = u'j1^i$ (1^i is the sequence of i "1") and, for every r, $r \leq i$, the form at occurrence $u'j1^r$ is α_r whilst $\alpha_1(\alpha_2\cdots(\alpha_i(\mathbf{f}(-)\cdots)$ is the j-th argument of a form at occurrence u'.

The following definitions are completely standard (see [Le78]).

Definition 5.5 The *degree* of a redex u in a term t (notation $\partial(u)$) is the label of the constructor (i.e. the sequence of the labels between the pair of symbols **d**-**c** of the redex u). ■

In general, if a redex u in a term t' has been obtained along a labelled derivation $\sigma : t \longrightarrow\!\!\!\!\!\rightarrow t'$ where $INIT(t)$ = true, we shall write $\partial_t(u)$ in order to emphasize the origin of the labelling.

Definition 5.6 Let t be a term in Σ^L. The predicate **INIT**(t) is true if and only if the labels of all forms in t are atomic and pairwise different from each other. ■

Definition 5.7 Let l be a label. The *norm* $\Delta(l)$ of l is inductively defined in the following way:

1. $\Delta(a) = 1 \quad (a \in L)$;
2. $\Delta(\ell_1\ell_2) = \Delta(\ell_1) + \Delta(\ell_2)$;
3. $\Delta((\ell)_s) = \Delta(\ell) + 1$. ■

Definition 5.8 For every labelled term t we call $\tau^{-1}(t)$ the corresponding (not labelled) term obtained by erasing all the labels in t. ■

Finally we denote with $\mathbf{ext}(t)$ the label of the outermost form in a term t.

Fact 5.9 Let t_1 be a redex. If $t_1 \rightarrow t_2$, then $\Delta(\mathbf{ext}(t_1)) < \Delta(\mathbf{ext}(t_2))$.

Proof: Let $\mathbf{ext}(t_1) = \alpha$ and let β the degree of the redex. Then, according to the definition of the rewriting rules in the labelled system, $\mathbf{ext}(t_2)$ is something of the form $\alpha(\beta)_0\gamma$. ■

The following theorems are the generalization of Theorems 2.5.3-4. (pp.47-48) in [Le78]. They essentially say that the labelling is powerful enough to discriminate all identifications of terms due to a syntactical coincidence (the typical example in the λ-calculus is $(\Delta\Delta)$ that reduces to "itself").

Theorem 5.10 For every pair of expressions t, t' such that $t \longrightarrow\!\!\!\!\!\rightarrow t'$, there exists a unique standard derivation between t and t'.

Proof: By the standardization theorem, we know the existence of at least one standard reduction $t \xrightarrow{\rho}\!\!\!\!\!\rightarrow t'$. Let l be the lenght of ρ and m be the size of (the syntactical tree of) t. The proof is by induction on $\langle l, m\rangle$.

(**base case**) Trivial, since t must be a variable.

(**general case**) By induction we know that the theorem is true for all pairs $\langle l', m'\rangle$ such that $l' < l$ or $l' = l$ and $m' < m$. Let us consider the following subcases according to the structure of t:
$(t = x)$ Trivial because the only reduction is the empty one.
$t = \mathbf{c}(\vec{x}^{\,1}.t_1, \cdots, \vec{x}^{\,n}.t_n))$ Then, $t' = \mathbf{c}(\vec{x}^{\,1}.t'_1, \cdots, \vec{x}^{\,n}.t'_n)$, $\rho = \rho_1 \ldots \rho_n$ and $t_i \xrightarrow{\rho_i}\!\!\!\!\!\rightarrow t'_i$. By induction, ρ_i is unique for every i, and so is ρ.
$(t = \mathbf{d}(t_0, \vec{x}^{\,1}.t_1, \cdots, \vec{x}^{\,n}.t_n))$ There are two subcases: either a redex involving $\mathbf{d}$ is fired in ρ or it is not. Moreover, all standard reductions from t to t' must be in the same class w.r.t. this partition since, by Fact 5.9, the external label of t' is different in the two cases.

If the destructor $\mathbf{d}$ is not involved in some reduction of ρ, we have a situation similar to the case when the most external form is a constructor. That is, $t' = \mathbf{d}(t'_0, \vec{x}^{\,1}.t'_1, \cdots, \vec{x}^{\,n}.t'_n)$, $\rho = \rho_0 \ldots \rho_n$ and $t_i \xrightarrow{\rho_i}\!\!\!\!\!\rightarrow t'_i$. By induction, ρ_i is unique for every i, and so is ρ.

In the other case, every standard derivation between t and t' is of the following form:

$$\mathbf{d}(t_0, \vec{x}^{\,1}.t_1, \cdots, \vec{x}^{\,n}.t_n) \longrightarrow\!\!\!\!\!\rightarrow \mathbf{d}(\alpha_1(\cdots(\alpha_i(\mathbf{c}(\vec{y}^{\,1}.t_1^+, \cdots, \vec{y}^{\,m}.t_m^+)\cdots), \vec{x}^{\,1}.t_1, \cdots, \vec{x}^{\,n}.t_n) \rightarrow t^+ \longrightarrow\!\!\!\!\!\rightarrow t'$$

The derivation

$$\mathbf{d}(t_0, \vec{x}^{\,1}.t_1, \cdots, \vec{x}^{\,n}.t_n) \longrightarrow\!\!\!\!\!\rightarrow \mathbf{d}(\alpha_1(\cdots(\alpha_i(\mathbf{c}(\vec{y}^{\,1}.t_1^+, \cdots, \vec{y}^{\,m}.t_m^+)\cdots), \vec{x}^{\,1}.t_1, \cdots, \vec{x}^{\,n}.t_n)$$

is the unique standard derivation internal to t_0 provided by the induction hypothesis. Thus every standard derivation in this class has the same prefix whose length is greater then 0. Now, by induction over the lenght of the derivation, also the standard reduction between t and t' is unique. ■

Theorem 5.11 Let t_L be a labelled term, and $t = \tau^{-1}(t_L)$. Let $t \xrightarrow{\sigma}\!\!\!\!\!\rightarrow t'$ and $t \xrightarrow{\rho}\!\!\!\!\!\rightarrow t''$ be two derivations in a *IS* (Σ, R) and $t_L \xrightarrow{\sigma_L}\!\!\!\!\!\rightarrow t'_L$ and $t_L \xrightarrow{\rho_L}\!\!\!\!\!\rightarrow t''_L$ be their isomorphic reductions in the labelled

calculus (Σ^L, R^L). Then $\rho \equiv \sigma \Leftrightarrow t'_L = t''_L$.
(Note that $\mathbf{INIT}(t_L)$ is not required).

Proof: ($\Rightarrow$) If $\rho \equiv \sigma$, $\rho_L \equiv \sigma_L$ since they are isomorphic. So $t'_L = t''_L$.

($\Leftarrow$) If $t'_L = t''_L$, by Theorem 5.10, there exists a unique standard derivation τ_L between t_L and t'_L and, by Theorem 3.9, $\sigma_L \equiv \tau_L \equiv \rho_L$. Thus $\sigma \equiv \tau \equiv \rho$, which by transitivity implies $\sigma \equiv \rho$. ■

6 λ-Discrete Interaction Systems

The problems in establishing a relation between zig-zag, extraction and labelling in Interaction Systems are essentially due to the fact that many elementary steps of substitution can be dealt with in a single rewriting rule. So the formal system is too abstract to allow the correct zig-zag. In the same way, extraction does not work since a given reduction may be usefull for the rest of the derivation in many different ways, each one inducing different canonical form. What is particularly problematic is the complex interplay between the creation of new forms and the substitution process. In a sense, we should provide a refinement of the IS, where each rewriting rule is splitted in several atomic steps: one step is that leading to the rhs where the substitutions are not computed yet; the other steps are those expanding these substitutions. An alternative but substantially equivalent approach is that of considering a subclass of IS's, where the substitution process is particularly simple and clear. The idea is simply that of reducing all binding operators to functional abstraction.

Definition 6.1 An Interaction Rule is *discrete* when the two interacting forms are discrete (i.e., they do not bind over their arguments). ■

Definition 6.2 A λ-DIS is an IS that contains only discrete interaction rules with the only exception of β-reduction. ■

In this way we have two completely separated level: new forms may be only introduced by discrete rules, while substitution can be only performed by β-reduction. Note that discrete rules may still create new lambda-terms, in the rhs.

Remark 6.3 Our λ-DIS are the counterpart of λ-TRS in [Kl80] (pp.135-137). ■

Any IS may be trivially simulated by a λ-DIS. In particular, every form $\mathbf{f}$ of arity $k_1 \ldots k_n$ is translated as a discrete form of arity 0^n (the sequence of n zeros). Bindings are dealt with by explicitly introducing λ-binders. For instance, a term of the kind $c(\langle x \rangle.M)$ will be translated as $c(\lambda(\langle x \rangle.M))$. Finally, in the rhs of the rewriting rules, wherever we have a subterm of the form $X[^M/_x]$, it will be "β-expanded" into an application XM.

We claim that the previous simulation of an IS into a λ-DIS is always "optimal" from the point of view os sharing. In other words, two redexes of an IS-term t are in a same family (have same degrees) if and only if the "corresponding" redexes in the associated λ-DIS are in a same family (have same degrees). The "natural" proof of this fact (i.e. by induction on the lenght of the derivation) is more

complex of what one could imagine. There are some technical points which are not easy to overcome, and we are still working at it. See also Remark 9.4 for a possible, alternative approach to the problem.

λ-DIS's have a particularly simple and clear theory of optimality, as we shall prove in the following sections. Indeed, in this case, we can still recover the nice correspondance between zig-zag, extraction and labelling, which holds for the λ-calculus.

We shall particularly concentrate our attention on the equivalence between extraction and labelling. We shall provide a completely original proof of this equivalence, avoiding the annoying notion of *labelled subcontext* [Le78].

The correspondance between zig-zag and labelling is exactly the same as in [Le80], so we shall skip it.

There is a final and important remark to make here. The solution of restricting the analysis to λ-DIS's has not been a choice "a priori". For quite a long time we hoped to provide a more exaustive account of the relations between labelling and extraction in the general case. Unfortunately we didn't succeed, and we have been essentially forced to go back to λ-DIS's (actually, we could make less restrictive hipothesis, but λ-DIS's are particularly clear and neat). This seems to comfort, also in the case of optimality, Curry's claim that "the theory of functional abstraction is tantamount to the theory of bound variables".

6.1 The extraction Relation

The extraction relation between redexes with histories was defined in [Le80] in the case of the λ-calculus. Every step of extraction eliminates a redex in ρu which do not take part in the creation of u. We recall here the definition (we shall use a slightly different notation, suggested by previous, more ambitious, attempts).

We say that two reductions are *disjoint* if they concern disjoint subterms of a term. Let $\mathbf{u}$ be a reduction $t \xrightarrow{u} t'$ of an *IS*. Therefore there exists a unique rewriting rule r_u whose instance corresponds to $\mathbf{u}$. Every metavariable X_i appearing in the lhs of r_u is matched by $\mathbf{u}$ with a subexpression t_i. According to our rule format, t_i's appear in the final expression t' as subexpressions $t_i[{}^{t'_1}/_{x_1}, \cdots, {}^{t'_n}/_{x_n}]$, for some n and expressions $t'_1, \cdots, t'_n$. The access path w of this subexpression will be called an *occurrence number of* X_i.

Definition 6.4 A reduction $\mathbf{v}_1; \cdots; \mathbf{v}_n$ starting at t' is *internal to the metavariable* X_i of $\mathbf{u}$ if there exists an occurrence number w of X_i such that, for every r and every k:

$$w \preceq v_r \quad \textbf{and} \quad \forall w^{(k)} \in o_{x_k}(t_i).\, ww^{(k)} \not\preceq v_i \tag{1}$$

where t_i is the subexpression of t replacing X_i and $x_1, \cdots, x_n$ are (the instances of) the bindings performed by the constructor or the destructor over the metavariable X_i.

Under the same hypothesis, $\mathbf{v}_1; \cdots; \mathbf{v}_n$ is *internal to the j-th occurrence* (from the left) *of* x_k in the metavariable X_i of $\mathbf{u}$ if there exists an occurrence number w of X_i such that, for every r:

$$v_r \succeq ww_j^{(k)}$$

where $w_j^{(k)}$ is the access path of the j-th occurrence (from the left) of the variable x_k in t_i. ■

When ρ is internal to the metavariable X_i of $\mathbf{u}$, we may lift it up before the firing of u, namely it is possible to perform the reduction ρ before the firing of $\mathbf{u}$.

Definition 6.5 Let ρ be internal to the metavariable X_i of $\mathbf{u}$ and let w ($w = uw^+$) be the occurrence number of X_i and w' be the access path of X_i in the lhs of the rewriting rule. Then the *lifting* of ρ w.r.t u (notation $\rho \uparrow_w^{w'} u$) is defined by replacing, inside every reduction of ρ, all the prefixes uw^+ with uw'.

We will often omit indexes w and w' (resp. w_j and w_1) in the notations of lifting. ■

Definition 6.6 The *extraction relation* $\rhd$ is the union of the following two relations:

1. $\rho;(\mathbf{u} \sqcup \sigma v) \rhd_1 (\rho;\sigma)v$ if $\mathbf{u}$ and $\sigma;\mathbf{v}$ are two *disjoint* reductions
2. $(\rho;\mathbf{u};\sigma)v \rhd_2^i (\rho;\sigma \uparrow_w^{w'} u)v'$ if $\sigma;\mathbf{v}$ is internal to the metavariable X_i of $\mathbf{u}$.

where, in 2, uw is the occurrence number of X_i, w' is the access path of X_i in the rewriting rule, $v = uwv''$ and $v' = uw'v''$.

We denote with $\trianglerighteq$ the reflexive and transitive closure of $\rhd$. ■

Lemma 6.7 The relation $\trianglerighteq$ has the Church-Rosser property, i.e. if $\rho \rhd \sigma$ and $\rho \rhd \tau$ then there exists a ν such that $\sigma \rhd \nu$ and $\tau \rhd \nu$.

Proof: By cases (m, n), according to the rule applied for $\sigma \rhd_m \nu$ and $\tau \rhd_n \nu$. The lacking cases (m, n) can be derived from the cases (n, m) with a simmetric reasoning.

(**case** (1, 1)) Let $\rho = \rho_1;(\mathbf{u} \sqcup (\rho_1;(\mathbf{v} \sqcup \rho_3)))$ such that both $(\rho_2;(\mathbf{u} \sqcup \rho_3))$ is disjoint with u and ρ_3 is disjoint with v. Thus let $\sigma = \rho_1;(\rho_2;(\mathbf{v} \sqcup \rho_3))$ and $\tau = \rho_1;(\mathbf{u} \sqcup (\rho_2;\rho_3))$. It is trivial to show that $\sigma \rhd_1 \rho_1;\rho_2;\rho_3$ and $\tau \rhd_2 \rho_1;\rho_2;\rho_3$.

(**case** (1, 2)) There are two subcases, according to the relative positions of the redexes.

(*Subcase a*) $\rho = \rho_1;(\mathbf{u} \sqcup \rho_2;\mathbf{v};\rho_3)$ and $\sigma = \rho_1;\rho_2;\mathbf{v};\rho_3$ and $\tau = \rho_1;(\mathbf{u} \sqcup \rho_2;\rho_3 \uparrow v)$. Then it is immediate that $\sigma \rhd_2 \rho_1;\rho_2;\rho_3 \uparrow v$ and $\tau \rhd_1 \rho_1;\rho_2;\rho_3 \uparrow v$.

(*Subcase b*) $\rho = \rho_1;\mathbf{u};\rho_2;(\mathbf{v} \sqcup \rho_3)$ and $\sigma = \rho_1;\mathbf{u};\rho_2;\rho_3$ and $\tau = \rho_1;(\rho_2;(\mathbf{v} \sqcup \rho_3)) \uparrow_w^{w'} u$. Notice that $\rho_2;\rho_3$ is a derivation which is still internal to the metavariable X_i of $\mathbf{u}$. Thus $\sigma \rhd_2^i \rho_1;((\rho_2;\rho_3) \uparrow_w^{w'} u)$. As far as τ is concerned, let $\rho_2 = \mathbf{u}_1;\cdots;\mathbf{u}_m$ ($u_i = uwu_i'$) and $\rho_3 = \mathbf{v}_1;\cdots;\mathbf{v}_n$ ($v_i = uwv_i'$). Then, by definition, $\tau = \rho_1;\rho_2^\star;(\mathbf{v}^\star \sqcup \rho_3^\star)$, where $\rho_2^\star = \mathbf{u}_1^\star;\cdots;\mathbf{u}_m^\star$ with $u_i^\star = uw'u_i'$ (similarly for $\rho_3^\star$ and $\mathbf{v}^\star$). Observe that the property to be disjoint between $\mathbf{v}$ and ρ is not changed after the operation of lifting. Thus $\tau \rhd_1 \rho_1;\rho_2^\star;\rho_3^\star$ and, it is trivial to check that $\rho_1;\rho_2^\star;\rho_3^\star$ coincides with $\rho_1;(\rho_2;\rho_3) \uparrow_w^{w'} u$.

(**case** (2, 2)) Let $\rho = \rho_1;\mathbf{u};\rho_2;\mathbf{v};\rho_3$ and $\sigma = \rho_1;(\rho_2;\mathbf{v};\rho_3) \uparrow_w^{w'} u$ and $\tau = \rho_1;\mathbf{u};\rho_2;\rho_3 \uparrow_z^{z'} v$ (thus $\rho_2;\mathbf{v};\rho_3$ is internal to (the occurrence uw of) the metavariable X_i in $\mathbf{u}$ and ρ_3 is internal to (the occurrence uwz of) the metavariable X_r in $\mathbf{v}$. Now, notice that $\rho_3 \uparrow v$ is still internal to the metavariable X_i in $\mathbf{u}$, thus $\tau \rhd_2^i \rho_1;(\rho_2;(\rho_3 \uparrow v)) \uparrow u$. On the other hand, again let $\rho_2 = \mathbf{u}_1;\cdots;\mathbf{u}_m$ ($u_i = uwu_i'$), $v = uwv'$ and $\rho_3 = \mathbf{v}_1;\cdots;\mathbf{v}_n$ ($v_i = uwv_i'$). Then $(\rho_2;\mathbf{v};\rho_3) \uparrow u = \rho_2^\star;\mathbf{v}^\star;\rho_3^\star$ where $\rho_2^\star = \mathbf{u}_1^\star;\cdots;\mathbf{u}_m^\star$ and $u_i^\star = uw'u_i'$ (similarly for $\mathbf{v}^\star$ and $\rho_3^\star$). Now, observe that ρ_3 is internal to the metavariable X_r

of $\mathbf{v}$, i.e. $v_i' = zv_i''$. Therefore $\tau \rhd_2^r \rho_1; \rho_2^\star; \rho_3^+$, where $\rho_3^+ = \mathbf{v}_1^+; \cdots; \mathbf{v}_n^+$ with $v_i^+ = uw'z'v_i''$. It is straightforward to check the coincidence of $\rho_1; \rho_2^\star; \rho_3^+$ and $\rho_1; (\rho_2; (\rho_3 \uparrow v)) \uparrow u$. ■

Now we have all we need for the definition of family.

Definition 6.8 A redex u with history ρ (notation ρu) is in the same *family* of a redex v with history σ, written $\rho u \approx \sigma u$, when there exists a τw such that $\rho_1 u \trianglerighteq \tau w \trianglelefteq \sigma_1 v$, where $\rho_1; \rho_2; \rho_3$ and $\sigma_1; \sigma_2; \sigma_3$ are respectively the standard derivations of ρ and σ and ρ_2 is an derivation internal to u and ρ_3 is disjoint to the right of u (similarly for σ_2 and σ_3 w.r.t. v). ■

Proposition 6.9 The family relation is effective.

Proof: As a matter of fact, the first step consisting in standardizing the derivation is effective by Theorem 3.9, the simplifications $\rhd_i$ are all trivially effective. By Lemma 6.7, it remains to show that $\rhd$ strongly normalizes. To this aim notice that $\rhd_1$ and $\rhd_2$ always shorten the derivation ρ in their lhs. ■

In the next section we will give more emphasis to the above proposition. We will show that our notion of family matches exactly with redexes having the same label, thus yielding a simple way for recognizing families.

Remark 6.10 The main differences w.r.t [Le80] are the following:

1. When ρ is internal to u, instead of our lifting, Lévy defines an operation $\rho||u$ in the following inductive way:

$$(\mathbf{v}; \rho)||u = v'/u; (\rho/V)||u \qquad \textbf{where } v \in v'/u \textbf{ and } V = v'/(\mathbf{u}; \mathbf{v})$$

   The derivation $\rho||u$ is thus the "completion" of ρ with all the redexes which are coresiduals of some redex in ρ w.r.t. the firing of u. Notice that $\rho \uparrow u$ gives a derivation which is coinitial with $\mathbf{u}; \rho$ whilst $\rho||u$ one gives a reduction which is coinitial with ρ. In the case of the λ-calculus, our lifting coincides with step 4 in the definition 4.7 of the extraction relation in [Le80]. Lévy's approach still work for λ-DIS, but it fails for more general systems (where we are still able to provide a corresponance between labelling and a suitable notion of extraction). This is essentially due to the possible nesting of two occurrences of a same metavariable. In particular, $\rho||u$ may now erase part of the original derivation ρ.

2. Lévy's definition of the extraction relation was based on foyr cases, while we only have three. This difference is more apparent than real. Case 1-2 of Definition 4.7 in [Le80] has been englobed in our first case. ■

7 Extraction preserves Labelling

Our aim is now to prove, in the case of λ-DIS's, the equivalence between the notion of family as defined by the extraction process, and that based on degrees. In this section we shall prove the easy part of this equivalence, namely that redexes in a same family have a same degree.

Theorem 7.1 Let $\rho u \approx \sigma v$, where $\rho : r \longrightarrow\!\!\!\!\!\rightarrow s$ and $\sigma : r \longrightarrow\!\!\!\!\!\rightarrow t$. Let $r^\star$ be an arbitrary labelling of r, and consider the isomorphic derivation $\rho^\star : r^\star \longrightarrow\!\!\!\!\!\rightarrow t^\star$ and $\sigma^\star : r^\star \longrightarrow\!\!\!\!\!\rightarrow t^\star$. Then the degree of (the redex $u^\star$ isomorphic to) u in $s^\star$ is the same of the degree of (the redex $v^\star$ isomorphic to) v in $t^\star$.

Proof: According to the definition, in order to prove that $\rho u \approx \sigma v$ we must first turn them into standard derivations, and then apply the extraction process to obtain a common canonical form. We shall prove that both operations do not modify the degrees of u and v in the corresponding labelled derivations.

Let us consider the derivation ρ.

(standardization) The standard derivation corresponding to ρ will be of the form $\rho_1;\rho_2;\rho_3$ where ρ_2 is internal to u and ρ_3 is disjoint to the right of u. Let $\rho_1^\star;\rho_2^\star;\rho_3^\star$ the isomorphic derivation in the labelled calculus. Note that, since $\rho^\star$ and $\rho_1^\star;\rho_2^\star;\rho_3^\star$ end in the same term, by Proposition 5.10, the degree of u cannot change in the standardization process. Now, since ρ_3 is disjoint to the right of u, it does not modify the degree of u. Hence the degree of u in $\rho^\star u$ and in $(\rho_1^\star;\rho_2^\star)u$ is the same. Similarly, since $\rho_2^\star$ is internal to u, there is no way for it to modify the degree of u, consequently u has the same degree both in $\rho^\star u$ and in $\rho_1^\star u$.

(extraction process) It remains to show that also the extraction relation $\trianglerighteq$ leaves unchanged the degree of u. It is enough to prove this fact for $\triangleright$. There are two cases:

$\triangleright_1$ Suppose that $\rho;(\mathbf{v} \sqcup \sigma u) \triangleright_1 (\rho;\sigma)u$. Then $\mathbf{v}$ and $\sigma\mathbf{u}$ must be disjoint reduction, and the firing of $\mathbf{v}$ cannot modify labels in the subterm concerning $\sigma\mathbf{u}$.

$\triangleright_2^{i,j}$ If $(\rho;\mathbf{v};\sigma)u \triangleright_2^{i,j} (\rho;(\sigma u) \uparrow v)$, the derivation $\sigma;\mathbf{u}$ is inside the j-th instance in the rhs of $\mathcal{R}_v$ of the i-th metavariable of its lhs. Again, the firing of $\mathbf{v}$ does not modify the labelling inside this instance. Since σu and $(\sigma u) \uparrow v$ execute the same reductions inside a same labelled term (only the context is different), the degree of the final redex must be the same. ■

8 Same degree implies same family

In this section we shall prove the difficoult part of the equivalence between extraction and labelling for λ-DIS's, namely that two redexes with a same degree are in a same family. Our proof is completely original; in particular we avoid to introduce the annoying notion of *labelled subcontext* (see [Le78]), by taking advantage of the new understanding of higher order calculi suggested by [Lam90] and [GAL92].

We shall start with providing an outline of the proof, that relies on two main lemmas. Then we shall briefly discuss the case of the λ-calculus, that is particularly simple and clear (due essentially to the fact that no new forms are created along a derivation). Then we shall come back to the general case.

8.1 The Main Theorem

Theorem 8.1 Let ρu and σv two redexes with coinitial histories, in canonical form w.r.t. $\trianglerighteq$. Let t be their common origin and consider the isomorphic labelled derivations in the hypothesis $\mathbf{INIT}(t)$.

Then

$$\rho u \neq \sigma v \;\Rightarrow\; \partial_t(u) \neq \partial_t(v)$$

The proof of the theorem is based on two main lemmas.

Definition 8.2 Let ℓ be a label. The set $\mathbf{at}(\ell)$ of atomic labels of ℓ is inductively defined as follows:

$$\begin{aligned} \mathbf{at}(a) &= a \\ \mathbf{at}(\ell_1\ell_2) &= \mathbf{at}(\ell_1) \cup \mathbf{at}(\ell_2) \\ \mathbf{at}((\ell)_s) &= \mathbf{at}(\ell) \end{aligned}$$

■

Lemma 8.3 Let t be such that $\mathbf{INIT}(t) = \mathit{true}$, $\sigma : t \Rightarrow t'$ be a standard derivation, v a redex in t' and σv in normal form w.r.t. $\trianglerighteq$. Let u be the leftmost outermost redex in t such that $\partial(u) \in \mathbf{at}(\partial_t(v))$. Then u is the first redex in $\sigma;\mathbf{v}$.

Lemma 8.4 Let $w\rho u$ and $w\sigma v$ be two canonical derivations w.r.t. $\trianglerighteq$, where $t \xrightarrow{w} t^+$. Then

$$\partial_{t^+}(u) \neq \partial_{t^+}(v) \;\Rightarrow\; \partial_t(u) \neq \partial_t(v)$$

By using the previous lemmas, the proof of theorem 8.1 is now an easy induction on the lenght of the shortest derivation between ρ and σ (assume it is ρ).

(**case** $\rho = \varepsilon$) Here u has to be a redex in the initial term t (thus its label is atomic). There are two subcases: $\sigma = \varepsilon$ and $\sigma = \mathbf{v}';\sigma'$. The first subcase is immediate because $u \neq v$ implies that their degree is different. In the second case, $\partial_t(v)$ cannot be atomic, since otherwise $\partial_t(v)$ would individuate a redex in t, and hence σv would not be in normal form w.r.t. $\trianglerighteq$.

(**case** $\rho = \mathbf{w};\rho'$) Let $\sigma = \mathbf{w}';\sigma'$. There are two subcases: either $w \neq w'$ or $w = w'$. In the first case, one between w and w' will be external to the other or they are disjoint. In both cases, by lemma 8.3 the leftmost outermost redex in $\mathbf{at}(\partial_t(u))$ and $\mathbf{at}(\partial_t(v))$ are different, and thus $\partial_t(u) \neq \partial_t(v)$.

If $w = w'$, we are in the hipothesis of lemma 8.4. Suppose $t \xrightarrow{w} t^+$. Since $\rho u \neq \sigma v$, we must have $\rho' u \neq \sigma' v$. By induction hipothesis, $\partial_{t^+}(u) \neq \partial_{t^+}(v)$ and, by lemma 8.4, $\partial_t(u) \neq \partial_t(v)$.

The next sections will be devoted to the proof of the two lemmas above. We shall start with discussing the case of the λ-calculus, which is particularly simple and clear. Then we shall discuss the general case.

8.2 The λ-calculus

The aim of this section is simply that of providing some intuition behind the proofs of Lemma 8.3 and 8.4 in the familiar case of the λ-calculus. The proofs (in particular that of Lemma 8.3) will be only sketched, here. For a deep analysys of the case of the λ-calculus, see [Lan92].

Proof of Lemma 8.3 The proof is by induction on the lenght of the derivation σ.

$\sigma = \varepsilon$ Obvious.

$\sigma = \mathbf{u};\sigma'$ Let $a = \partial_t(u)$, and $u = (\lambda x.M)N$. We distinguish two cases.

1. $\sigma';\mathbf{v} = \sigma_1;\mathbf{u}';\sigma_2$, where σ_1 is internal to $M[N/x]$, and $\mathbf{u}'$ is a redex outside $M[N/x]$. In this case, the redex $\mathbf{u}$ was inside an application, i.e. we were in a situation of the kind $((\lambda x.M)N)Q$. Moreover $\sigma_1 : M[N/x] \longrightarrow\!\!\!\!\!\rightarrow \lambda z.P$, and $\mathbf{u}' = (\lambda z.P)Q$. Note now that the label of the subterm $\lambda z.P$ eventually starts with $\overline{a}$, and this label is captured in the degree of u'. Then we use the induction hypothesis on the derivation $\mathbf{u}';\sigma_2$.
2. The other case is when the derivation $\sigma' v$ is internal to u (otherwise it is easily proved that σv could not be canonical w.r.t. $\trianglerighteq$). Let $u = (\lambda x.M)N$. The derivation $\sigma' v$ cannot be internal to M or N, since otherwise u could be removed from σ. This means that in $\sigma' v$ there exist a redex w' "on the border" between M and N (more precisely, whose application node comes from M and whose λ-node comes from N). This means that the derivation $\sigma';\mathbf{v}$ will be of the form $\sigma_1;\sigma_2;w';\sigma_3$ where σ_1 reduces redexes in M, and σ_2 is internal to N. Now use the induction hypothesis on $w';\sigma_3$, and observe that, when we start with $\mathbf{INIT}(t)$, $\partial(w')$ eventually contains $\underline{a}$. ■

Let us come to the proof of the second lemma 8.4 (we will only provide some hints, since it will be considered in full details in the general case).

For this purpose, it is convenient to introduce the notion of *path* associated with a label. Consider a term t such that $\mathbf{INIT}(t)$. Every edge in t is labelled with a different atomic symbol, so we may call each edge by its label. We shall assume here the graph representation of terms; i.e. we suppose that bounded variables are explicitly connected to the respective binders. If l is a label of an edge generated along some reduction from t, we define the path of l in t in the following way:

$$\begin{aligned} path(a) &= a \\ \mathbf{path}(\ell_1\ell_2) &= \mathbf{path}(\ell_1)\cdot\mathbf{path}(\ell_2) \\ \mathbf{path}(\overline{\ell}) &= \mathbf{path}(\ell) \\ \mathbf{path}(\underline{\ell}) &= (\mathbf{path}(\ell))^r \end{aligned}$$

where "$\cdot$" means concatenation, and $(p)^r$ is as p but reverted.

It is easy to proof by induction on the lenght of the derivation generating the label l that the previous definition is sound, i.e. that $\mathbf{path}(l)$ is indeed a path in t.

The main idea behind the notion of path, is that the a label l on a given edge q is obtained by "contraction" of $\mathbf{path}(l)$; this contraction may be only realized by firing redexes which are along the path.

Remark 8.5 As far as we know, the understanding of a computation as a "travel along a path" has been pointed out for the first time by Danos and Regnier (personal communication). The notion of (consistent) path has been formalized in [GAL92] (warning: you will have to deal with the "terrific" *bus-notation*). In the same paper, you will also find a discussion of the relations between paths and Lévy's labelling. ■

The function **path** is injective other degrees. In other words, there exists a unique standard way to "contract" a path, in order to generate a label. In order to prove this important theorem, a preliminary result is required, concerning the structure of labels.

Proposition 8.6 Let $\ell = \alpha_1 \ldots \alpha_n$ be a label generated along some derivation σ, where each α_i is a proper label. Then the sequence $\alpha_1 \ldots \alpha_n$ is alternately composed by atomic labels and underlined or overlined labels. Moreover,

1. α_1 and α_n are always atomic;
2. if $\alpha_i = \overline{\ell_1}$, then α_{i-1} is an atomic label relative to some (subterm starting with) @ in the original term.
3. if $\alpha_i = \underline{\ell_1}$, then α_{i-1} is an atomic label relative to a bounded variable in the original term.

Proof A trivial induction on the lenght of the derivation. ■

Note in particular that, if ℓ is a label relative to a redex, it may appear inside another label only if it is overlined or underlined, and with an atomic label at its left (and right).

Proposition 8.7 The function **path** is injective over labels generated along derivations starting from an initial labelling. Moreover, a path relative to a redex cannot be an initial subpath of any other path associated to a different label.

Proof: Let ℓ_1 and ℓ_2 two different lables. The proof is by induction on the structure of ℓ_1.

1. ℓ_1 is atomic. Trivial.
2. $\ell_1 = \alpha_1 \ldots \alpha_n$. If $\ell_1 = \ell_2 \ell$ (or viceversa) the thesis follows trivially, since the two paths have different lenghts. Note that this case is not possible when one of the two labels ℓ_1 or ℓ_2 is relative to a redex, by the remark after Proposition 8.6. Suppose then $\ell_2 = \beta_1 \ldots \beta_m$, and let k be the first index such that $\alpha_k \neq \beta_k$. Three subcases are possible.
   - α_k is atomic. By Proposition 8.6, also β_k must be atomic. Since they are different, the two paths diverge here.
   - $\alpha_k = \overline{\ell_1'}$. Then $\beta_k = \overline{\ell_2'}$ by Proposition 8.6 (note that β_k cannot be underlined, since otherwise, $\alpha_{k-1} \neq \beta_{k-1}$). Then we use the inductive hypothesis over ℓ_1' and ℓ_2' (note in particular that the two paths must really diverge and they cannot be one an initial subpath of the other).
   - $\alpha_k = \underline{\ell_1'}$. Analogous to the previous one.

■

Using the notion of path, lemma 8.4 is then proved as follows. Consider the two derivations $w\rho u$ and $w\sigma v$ in canonical form w.r.t. $\trianglerighteq$, where $t = T[(\lambda x.M)N] \xrightarrow{w} t^+ = T[M[^N/_x]]$. The hipothesis, $\partial_{t^+}(u) \neq \partial_{t^+}(v)$. This means that $\partial_{t^+}(u)$ and $\partial_{t^+}(v)$ define two different paths p_1 and p_2 in t^+. Now,

the only possibility to get exactly the same labels along p_1 and p_2 when we start with $\mathbf{INIT}(t)$, is that p_1 and p_2 are internal to two different instances of the argument N (this is proved by a boring and tedious case analysis). Since this case is excluded by the hipothesis that $w\rho u$ and $w\sigma v$ are in canonical form w.r.t. $\trianglerighteq$, we must eventually have $\partial_t(u) \neq \partial_t(v)$.

8.3 λ-DIS's

Lemma 8.3 has a straighforward generalization to λ-DIS's. Actually, it also holds for Interaction Systems, and we shall prove it in this general case.

General Proof of Lemma 8.3 The proof is by induction on the lenght of σ.

$\sigma = \varepsilon$ Obvious.

$\sigma = \mathbf{u}; \sigma'$ Let a be the degree of u in $\mathbf{INIT}(t)$. Let w be the first redex in $\sigma' v$. We distinguish two main cases.

1. $\sigma'; \mathbf{v} = \mathbf{w}; \sigma_1; \mathbf{u}'; \sigma_2$, where $\mathbf{w}; \sigma_1$ is internal to the rhs of the reduction $\mathbf{u}$ and u' is a redex outside the rhs of $\mathbf{u}$.

   Let $t \xrightarrow{u;w;\sigma_1} t'$. By induction hypothesis, $\partial_{t'}(u) \in \mathbf{at}(\partial_{t'}(v))$. But, $a \in \mathbf{at}(\partial_t(u'))$, since the the outermost form of the subexpression at u must be involved in the reduction u' (otherwise σv could not be standard or in normal form w.r.t $\trianglerighteq$). Then $a \in \mathbf{at}(\partial_t(v))$ (this kind of "transitivity" can be easily formalized by considering graph morphisms between labelled terms).

2. $\sigma'; \mathbf{v}$ is internal to the rhs of the reduction $\mathbf{u}$.

   Let H be the *smallest* subexpression in the rhs of the reduction $\mathbf{u}$ containing $\sigma'; \mathbf{v}$. The proof is by cases on the possible structure of H.

   (**case** $H = x$ **or** $H = \mathbf{c}$) (the arity of $\mathbf{c}$ is empty) They are vacuous because there exists no reduction $\mathbf{w}$ inside H.

   (**case** $H = \mathbf{f}(\vec{x}^1.H_1, \cdots, \vec{x}^n.H_n)$). $\mathbf{f}$ must be a destructor, since otherwise $\sigma' v$ would be internal to some H_i. In particular, a (strict) prefix of $\sigma' v$ is internal to the first argument of $\mathbf{f}$ till a constructor $\mathbf{g}$ is produced such that $\mathbf{f}$-$\mathbf{g}$ can be fired. But such redex, according to our labelled rules, has eventually a degree containing $\partial_t(u)$. Thus we get the thesis by resoning as in case 1.

   (**case** $H = X[{}^{H_1}/_{x_1}, \cdots, {}^{H_n}/_{x_n}]$). Let w be the first redex fired in $\sigma'; \mathbf{v}$. The case in which w is internal to some H_i is proved as in case 1 ($\sigma'; \mathbf{v}$ eventually creates redexes until we exit from H_i towards its "root").

   The interesting case is when w is a redex of X. In general the reduction $\sigma'; \mathbf{v}$ will be of the form $\sigma_1; \sigma_2; \mathbf{u}'; \sigma_3$ where σ_1 reduces redexes in X and σ_2 is a derivation internal to some instance of H_i and u' is the first redex not internal to that instance.

   We prove that the degree of u' contains $\partial_t(u)$ (and thus we can run again the reasoning in **(A)** for proving the theorem). Of course, $\sigma_2; \mathbf{u}'; \sigma_3$ must be a canonical derivation otherwise $\sigma' v$ should be internal to X, contrarily to the hypothesis of normal form w.r.t. $\trianglerighteq$ for σv.

Now the critical observation is that all the instances of H_i (for every i) can be duplicated or deleted along σ_1 but they remain disjoint. This because after $\mathbf{u}$ such instances are trivially disjoint and a standard induction over the length of σ_1 shows that this property is preserved by firing redexes in X.

Thus if u' was a redex in another instance of H_i or in an instance of H_j, $j \neq i$, then a contradiction is easily raised. Actually $\mathbf{u}'$ must be disjoint "at the right" of σ_2 and the tailing $\mathbf{u}'; \sigma_3$ can not fire redexes involving subexpressions "disjoint to the left" (because σv is standard). Therefore, we can eventually simplify σv according to $\trianglerighteq$, contrarily to the hypothesis.

Consequently, the unique case is that u' is external to the instance of H_i. Here the outermost symbol of H_i must be involved in the reduction otherwise the redex was ready to be fired before σ_2. The thesis follows because the label of the outermost form of H_i contains $\partial_t(u)$.

■

On the contrary, the proof of lemma 8.3 is much more complex than in the case of the λ-calculus. The main problem is that we cannot rely anymore over the simple notion of path associated with a label, as in the case of the lambda calculus. Indeed, discrete rules allow us to create new operators along a derivations, and the flow of information along these forms is not captured by a path in the initial term.

So we shall try to use the intuition provided by paths, without explicitly mentioning them. For this purpose, we need a much closer inspection of labels and their "structure". We shall discover in this way that there is a lot of redundancy in Lévy's labelling (and our generalization), just due to the fact that a label describes a travel along a connected "path".

Our first aim is that to formally capture the "abstract structure" of a label. This is done by introducing a weaker form of initial labelling.

Consider the following set of atomic labels

$$L^\bullet = \{\bullet\} \cup \{\bullet_i \mid i \in \omega\}$$

Labels $\mathbf{L}^\bullet$ are defined in the usual way. Given a term t labelled by elements in $\mathbf{L}^\bullet$, we say that $\mathbf{INIT}^\bullet(t)$ is *true* when every arc joining two free ports is labelled by $\bullet$ and incoming arcs into bounded ports are labelled by $\bullet_n$ (for some n) with the constraint that, given a bounded port, an incoming edge has n as index of bullet iff it is the n-th incoming edge in that port from the left (thus, given a port in t, different incoming edges have different indexes of bullets).

Definition 8.8 Let $\sigma : t \twoheadrightarrow t'$ be a derivation, and let σ_I and $\sigma^\bullet$ be the isomorphic labelled derivations starting with $\mathbf{INIT}(t)$ and $\mathbf{INIT}^\bullet(t)$, respectively. Let u be an edge in t' and ℓ be the label got by u along σ_I. The *structure* $\mathbf{str}(\ell)$ of ℓ is the label got by the same edge u along $\sigma^\bullet$. ■

Note that the definition is sound, i.e. it only depends from the label, and not from the edge (i.e. same labels on different edges have the same structure). We may equivalently define the $\mathbf{str}(\ell)$ by an obvious substitution for atomic symbols from L to $L^\bullet$. The equivalence between the two definitions is a straighforward induction on the lenght of σ.

Now, we have a surprising result (this result holds for arbitrary IS's).

Theorem 8.9 Let t be a term such that **INIT**(t), and let ℓ be a label generated along some labelled reduction σ. Then ℓ is uniquely determined by its structure **str**(ℓ) and a single atomic label of L in a specified occurrence inside s(ℓ).

The proof of this result is quite technical, and it is reported in appendix B. However, the idea is very simple: given an occurrence of an atomic symbol inside $s(\ell)$, we uniquely determine all the other symbols "by connection".

A preliminary lemma is required.

Lemma 8.10 Let $w\rho u$ and $w\sigma v$ be two canonical derivations w.r.t. $\trianglerighteq$, where $t \xrightarrow{w} t^+$. Then

$$\partial_t(u) = \partial_t(v) \Rightarrow \mathbf{str}(partial_{t^+}(u)) = \mathbf{str}(\partial_{t^+}(v))$$

Proof: Let a be the degree of w in $INIT(t)$. Consider all the occurrences of a inside $\partial_t(u) = \partial_t(v)$. The structure of $partial_{t^+}(u))$ and $\partial_{t^+}(v)$ is obtained from the structure of $\partial_t(u) = \partial_t(v)$ by a suitable "local modifications" around the occurrences of a. Let us consider the two cases when w is a β-reduction or a discrete rule (for the sake of clarity we shall use overlinings and underlinings, in the case of β-reductions).

1. Suppose $w = @(\lambda(\langle x\rangle.M), N)$. No label of free variables in M or N may appear in $\partial_t(u)$, since this would imply firing a redex external to w, while the derivation $w\rho u$ is standard (note that this is not true in general IS's, anf this fact creates a lot of troubles). Then, we have just to replace every sublabel of the kind $\bullet\overline{a}\bullet$ and $\bullet_n\underline{a}\bullet$ with a single $\bullet$. The case when M is the identity is particular: in this case we shall find sublabels of the kind $\bullet\overline{a}\bullet_1\underline{a}\bullet$ which must be replaced with a single $\bullet$.

2. Suppose w is a discrete reduction. In this case we have just to peplace all sublabels $(a)_s$ not relative to variables (this information is provided by s, by inspection of the rewriting rule R associate with w) with $\bullet$. As for the other $(a)_s$, they should be replaced with $\bullet_n$, where again n is a function of s and R (we have other problems, here, in the case of general IS's).

■

Let us come to the proof of our second main lemma (for a more formal and direct proof, see [Lan92]). We recall the statement.

Lemma 8.4 Let $w\rho u$ and $w\sigma v$ be two canonical derivations w.r.t. $\trianglerighteq$, where $t \xrightarrow{w} t^+$. Then

$$\partial_{t^+}(u) \neq \partial_{t^+}(v) \Rightarrow \partial_t(u) \neq \partial_t(v)$$

Proof: By lemma 8.10, $\partial_{t^+}(u)$ and $\partial_{t^+}(v)$ must have the same structure. Then the idea of the proof is the following. We will choose *in a suitable way* an occurrence b of an atomic label in $\partial_{t^+}(u)$ or $\partial_{t^+}(v)$, and we shall consider the corrisponding label b' in the other degree (this is well defined, since the two degrees have the same structure). We cannot have $b = b'$, since otherwise by theorem B.1 the

two degrees would be equal. So $b \neq b'$, and all we have to prove is that the edges e and e' correspondig to b and b' in t^+ get different labels when starting from $INIT(t)$.

The choice of b is not difficoult. Let R be the rewriting rule associated with w. Consider the derivation ρu. It cannot be internal to some instance of some metavariable in the lhs of R, since $w\rho u$ is canonical. This means that in $\partial_{t^+}(u)$ will appear at least one atomic label external to all instances of metavariables in the lhs of R. This is our b. Note now that all edges of t^+ wich are external to instances of arguments of w get different labels through the firing of w, and we are done. ■

9 About Implementation

We shall briefly discuss, in this section, the optimal implementation of IS's by means of Lamping-Gonthier's sharing and control operators. Our aim, here, is not that of providing a complete account of this topic (that will eventually be the subject of the forthcoming Part II), but just to stress the relevance of IS's from this operational point of view. Actually, IS's have been primarily inspired by the need of extending (in a uniform way) Lamping's implementation technique to more constructs than just β-reduction.

The general idea behind the implementation is pretty simple. We have already remarked that there exists a "logical" intuitionistic system L associated with every IS (we may suppose to describe L in sequent style). Then, destructors correspond to left introduction rules, and constructors to right introduction rules (reduction rules describe the process of cut-elimination). All the other rules of the logical system L are the usual structural rules of intuitionistic logic.

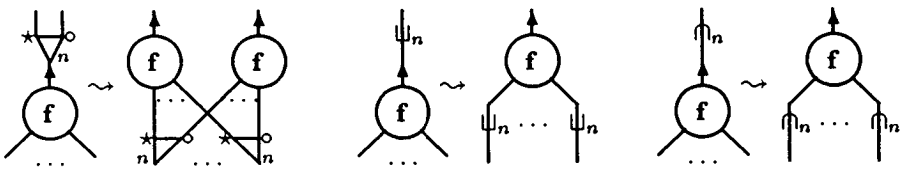
Now, we can define a "linear logic" version of L. That is, we may define a new system L_{LL} by just replacing the structural part of L with its "counterpart" in Linear Logic (using a monoidal product and the comonad "!"). Then, L can be embedded into L_{LL} in essentially the same way that Intuitionistic Logic is embedded into (Intuitionistic) Linear Logic. Finally, we use the optimal implementation of *boxes* defined in [GAL92] to get an optimal implementation of the original IS's.

We will not have much space, in this extended abstract, to discuss neither the correctness nor the optimality of our implementation. However, by the previous discussion, it *intuitively* follows from the correctness of the local, optimal implementation of *boxes* described in [GAL92].

The rewriting sistem which describes the optimal implementation of an IS is still an Interaction Net in the sense of Lafont [Laf90]. The translation of expressions is discussed in Subsection 9.1 below. Rules may be classified in three groups.

(control rules) These are the 12 rules in [GAL92, GAL92], which describe the operational behaviour of the control operators: fans, croissants and brackets (we shall use the symbol $\cup$ instead of the croissant).

(interfacing rules) These are the rules which describe the interaction between control operators and forms of the syntax. We have three classes of rules for every form $\mathbf{f}$ (n is any integer greater than 0):



(**proper rules**) These rules describe the interaction between destructors and constructors. These are the only rules which are dependent from the particular Interaction System under investigation. The translation of proper rules will be the subject of Subsection 9.2.

9.1 The translation of expressions

We will describe in this section the translation of arbitrary terms in sharing graphs. We shall essentially follow [GAL92], that is particularly clear, even if a bit redundant.

For simplicity, we shall only consider the paradigmatic case when constructors and destructors have respectively arity 1 and 01. The other cases are easily derived. The translation is a function $\mathcal{T}$ which is inductively defined in Figure 3.

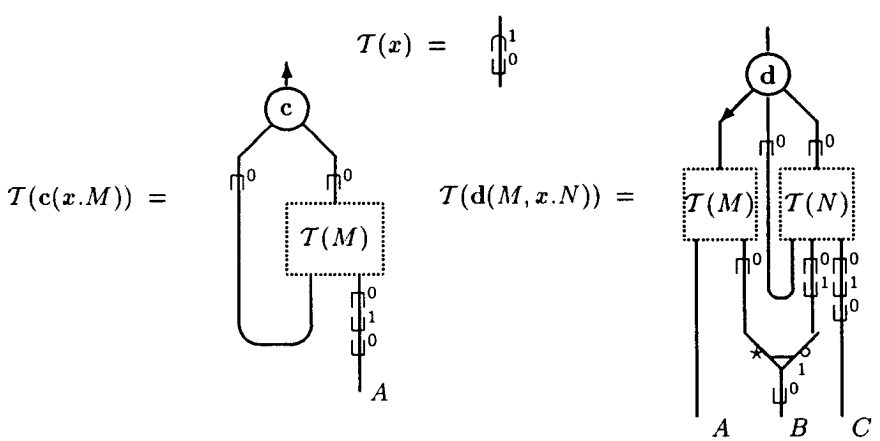


Figure 3: The translation function $\mathcal{T}$

In this figure the edges labelled A, B and C represent generic free variables which are not bound by the forms **c** and **d**. In particular, B represents a free variable which is common to M and N.

Moreover, ports of a form **f** leading to a bound variable not appearing in the argument are terminated by a plug, in the usual way.

9.2 The translation of rewriting rules

Let us come to the implementation of the rewriting rules. For the sake of clarity, we shall define it in several steps.

Let R be an IS rewriting rule. First of all, we must put the rhs of R in a suitable form, in order to emphasize the "interface" between the new structure introduced in the rhs and the metavariables in

its lhs.

(**β-expansion**) The first step is to β-expand all substitutions in the rhs. For this purpose we shall use two new *pseudo* operators of abstraction **Abs** and application **App** (with the usual meaning: these are not to be confused with other forms of the syntax).

Example 9.1 Consider the rewriting rule for μ:

$$\mu(\lambda(x.X)) \rightarrow X[^{\mu(\lambda(y.X[^{y}/_{x}]))}/_{x}]$$

The β-expansion of the rhs gives the following term:

$$\mathbf{App}(\mathbf{Abs}(x.X),\ \mu(\lambda(y.\mathbf{App}(\mathbf{Abs}(x.X),\ y))))$$ ■

Note that all occurrences of metavariables are now closed with pseudo binders, i.e. they are of the form $\mathbf{Abs}(x.X)$.

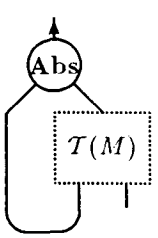
(**linearization**) After the β-expansion, all metavariables X appear in expressions of the form $\mathbf{Abs}(x.X)$. The next step is to make the rhs *linear* w.r.t. such occurrences. This is obtained by replacing all occurrences of $\mathbf{Abs}(x.X)$ with a new fresh variable w, by abstracting over w with a further pseudo abstraction, and by passing $\mathbf{Abs}(x.X)$ as an argument to this abstraction.

Example 9.2 After the linearization step, the rhs of the recursion rule becomes:

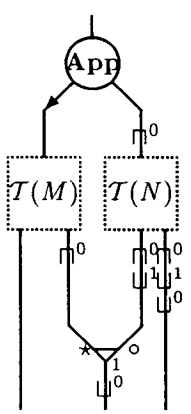
$$\mathbf{App}(\mathbf{Abs}(w.\mathbf{App}(w,\ \mu(\lambda(y.\mathbf{App}(w,\ y))))),\ \mathbf{Abs}(x.X))$$ ■

(**translation**) Now we are ready to translate the rhs. Forms of the syntax are translated as described above. For pseudo abstraction and pseudo application we use the usual (call by name) translation, described by the following rules:

$\mathcal{T}(\mathbf{Abs}(x.M)) =$



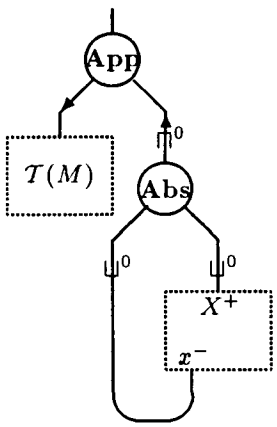
$\mathcal{T}(\mathbf{App}(M,\ N)) =$



Observe the absence of brackets around the argument of **Abs**. This because, in ordinary β-reduction, there is a linear use of the functional body M of the abstraction.

Now, the essential fact is that, during the translation, we may consider each subterm $\mathbf{Abs}(x.X)$ as a "black-box". These subterms will eventually appear in a context of the kind $\mathbf{App}(M,\ \mathbf{Abs}(x.X))$. In this case, instead of the above rules, we have the following simple translation:

$\mathcal{T}(\mathbf{App}(M,\ \mathbf{Abs}(x.X))) =$

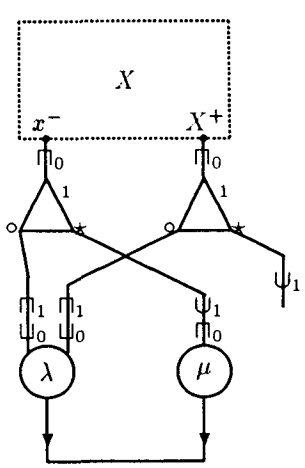


Indeed, the instance of X has been already put inside a box during the translation of terms, and we use this box instead of building a new box around the argument of the application. In this way, no operation around the (unaccessible!) free variables of the instance of X must be performed. Moreover, we do not have problems with sharing since all variables in instances of X are eventually different from all the new variables introduced in the rhs of the rule (and sharing of variables among instances of different metavariables is already expressed).

The final step is to partially evaluate the term we have obtained after the translation w.r.t. all pseudo operators. For the conditions on the rewriting rules of an IS, it is possible to prove that all pseudo operators may be reduced in this way.

Recall that the reduction rule for pseudo application and abstraction is

Example 9.3 By applying the previous technique (and some optimizations not worth discussing here) we obtain the following implemetation of the rhs of the rule concernig μ:



Remark 9.4 Our implementation of an IS by means of sharing graphs essentially passes through its preliminary translation as a λ-DIS's (β-expansion and linearization). The main difference is that the λ-binders simulating the abstraction of forms are not introduced at "compile time", but at "run time", namely when a rule involving those forms is reduced. This is done in order to partially evaluate the rewriting rule w.r.t. all its "pseudo" operators. As a consequence, if we prove that the previous implementation is "optimal" w.r.t. our general labelling for IS's, we have implicitly proved our claim of section 5, namely that the simulation of an IS as a λ-DIS is optimal from the point of view of sharing. ■

The previous translation could (and must) be improved. Apart from studying optimization techinques for reducing the number of sharing operators, the translation should be *relativized* to the particular IS's under investigation. In particular, some operators of the syntax could make only a linear use of some of their arguments. For instance, this is the case of the λ-calculus, where the body of the abstraction is treated linearly in β-reduction. These linear arguments have a simpler translation, since there is no need to put them inside a "box". Furthermore, remark that the decision that some operator **f** behaves linearly over an argument needs the examination of *all* the interaction rules involving **f**.

10 About correctness and optimality

Correctness can be essentially proved by following the same techniques in [Lam89] and [GAL92, GAL92], based on the so called context semantics. Lamping's original technique seems particularly easy to generalize (and it could be even simplified by our handling of bound variables). The approach in [GAL92, GAL92] is slightly different, since also the forms of the syntax are considered as an integrating part of the context semantics, providing in this way a more elegant and comprehensive "semantical" description. This is particularly simple in the case of the λ-calculus, since abstraction and application can be assimilated to fan-nodes. In our case, we should introduce new control marks apart from $\circ$ and $\star$ (actually, a different mark for every auxiliary port of each form). Again, however, this does not seem to be particularly problematic.

The problem of optimality looks more complex. The first step is to fix a correspondence between expressions of a labelled IS (which, in turn, have a graphical representation in the same style of Section 3.1) and "labelled" sharing graphs. Reduction rules are extended in order to make labels commute with all control nodes. Finally a labelled reduction in the evaluator is defined in the same way as the corresponding contraction in the graphical representation of Section 3.1. At this point, the optimality of the implementation is expressed by the following theorem.

Theorem 10.1 In every derivation no two redexes destructor-constructor have the same label. Therefore the graph implementation is optimal. ■

Both the proof in [Lam90] and [GAL92] are essentially based on the notion of *prerequisite chain* of a form **f**. Roughly, a prerequisite chain for **f** is a path in the initial term starting from **f** and traversing those redexes which are to be contracted in order to yield a redex involving **f**. For Lamping, the "path" is not connected, since bound variables are not connected to their binders. In [GAL92] the notion of

prerequisite chain is replaced by the simpler notion of *legal path* between abstractions and applications (rightmost fans). Legal paths are simply defined according to the context semantics.

Now, our main problem is that, due to the creation of new forms along a reduction, a *prerequisite chain* of a given form cannot be considered any more as a "path" in the initial term. The problem is essentially notational, but all the proofs get enourmously entangled, and we are still checking them.

11 Conclusions

We have introduced in this paper a new class or rewriting systems, called Interaction Systems. IS's are the intuitionistic generalization of Lafont's Interaction Nets. In particular, we keep the idea of binary interaction, and the syntactical bipartition of operators into *constructors* and *destructors*.

IS's are subclass of Klop's Combinatory Reduction Systems. More specifically, they are the subclass of CRS's where the Curry-Howard analogy between proofs and terms still makes sense. This means that we can associate to every IS a suitable "logical" (intuitionistic) system: constructors and destructors respectively correspond to right and left introduction rules, interaction is cut, and computation is cut-elimination.

We have investigated in this paper the theoretical aspects of Interaction Systems related to the problem of optimal reductions. We have defined an extension of Lévy's labelling to arbitrary IS's, and we have put in evidence the problems of relating labelling to other standard definitions of the family relation (zig-zag, and extraction). In particular, we have proved the inadequacy of Lévy's zig-zag.

We have then restricted our attention to a subclass of IS's, called λ-DIS, where the theory of optimality looks particularly simple. In this case, we have been able to recover the equivalence between labelling and extraction. Our proof is completely original, and it looks simpler than Lévy's original proof [Le78].

Finally, we have briefly discussed the optimal implemetation of IS's by using Lamping's graph reduction technique (IS's have been primarily inspired by this impementation issue).

A lot of work is left. We shall list below the main open problems suggested by this paper.

11.1 Implementation

We are at a good point in proving correctness and optimality of our implementation, but a lot of work should be still devoted to refining techniques and terminology. In our experience, proofs looked obsolete as soon as we completed them. We made a lot of effort in searching a good compromise between the (alas, opposite) requirement of being clear and of being formal; we are not sure of having succeed.

It would be particularly important to extend the *bus notation* ([GAL92]) to IS's. This notation is particularly appealing for an actual implementation, since it enormously reduces the number of "mutual crossing" between sharing operators. Once we have a bus-notation, also the elegant *context semantics* of [GAL92] should have a natural generalization, which in turn could simplify the correctness proof of the implementation.

We are also working at the generalization of the *read-back procedure* as defined in [As91] in the case of the λ-calculus.

However, the main problem that is still open (even in the case of the λ-calculus) is that to study (and possibly avoid) the proliferation of control operators (this problem is particularly clear from the categorical perspective proposed in [As91]).

11.2 Syntax

There are some interesting syntactical aspects of IS's which are worth to be investigated. In particular, IS's are *locally sequential*: the forms have a main port, and interaction (reduction) is only allowed at this position. This suggests that should be possible to generalize Berry's theorem [Be78] from λ-calculus to arbitrary IS's (that, in turn, would require the generalization of Bohm-trees [Ba77]).

A different class of problems is posed by possible syntactical extensions of IS's. The case of unary interaction seems particulary easy to solve, since it can be considered as a particular case of binary interaction. The typical example is the recursion operator μ, that is usually described by the following rewriting rule:

$$\mu(\langle x\rangle.X) \;\rightarrow\; X[^{\mu(\langle x\rangle.X)}/_{x}]$$

There are at least two "standard" ways to code such forms in Interaction Systems. The first is to consider them as constructors and to introduce dummy destructors of arity 0 interacting with them. Thus, in the case of μ, we take a destructor $\mathbf{d}_\mu$ and the rewriting rule becomes:

$$\mathbf{d}_\mu(\mu(\langle x\rangle.X)) \;\rightarrow\; X[^{\mathbf{d}_\mu(\mu(\langle x\rangle.X))}/_{x}]$$

The other way consists of considering operators interacting unarily as destructors and require the existence of "dual" dummy constructors of arity ε. Again, when the μ is considered, the dummy constructor is $\mathbf{c}_\mu$ and the rewriting rule becomes:

$$\mu(\mathbf{c}_\mu, \langle x\rangle.X) \;\rightarrow\; X[^{\mu(\mathbf{c}_\mu,\langle x\rangle.X)}/_{x}]$$

Both solutions are equivalent from the point of view of optimality. Of course, there is no need of explicitly introduce the dummy operators in a real implementation; they just help to get the right internal representation. Note also that both the previous "standard" encoding of the μ operator are different from that adopted along the paper, that was essentially choosen for dydactical reasons.

On the contrary, more complex kinds of interactions immediately create problems. Let us consider the general case of CRS's. The first problem is that each form may work as a constructor or a destructor, depending from the context. This means that the decision if a control operator may or may not travers $\mathbf{f}$ is not local any more. In other words, interfacing operators of the syntax with control operators, that is so simple in the case of IS's, becomes a terrible task in CRS's (in particular, this class of the rewriting rules will not be, in general, in the form of Interaction Nets). The second problem is to decide where to put boxes, and when we may safely open them. Again, we had a more or less trivial solution in the case of IS's, essentially due to their *logical* nature. Finally, there is an even subtler problem. If the lhs of rules has a complex shape, as in the case of CRS's, the term it

has to be matched against may contains shared subterms. So, from one side the matching operation may become quite complex, and on the other side we must correctly keep the old sharing after the reduction of the rule!

All this looks very complicate, and it seems to suggest that IS's are really the right setting for investigating the problem of optimal implementation of higher order rewriting systems.

11.3 Redex families

An interesting problem is posed by the zig-zag relation. This approach to the definition of the family relation was particularly appealing for its abstract nature. Indeed it is entirely based on the notions of residual and permutation equivalence, without any reference to the underling syntax of the particular language under consideration. Unfortunately, and quite surprisingly, it is not adequate. This suggests at least two different problems to be solved. First of all, we should look for the "correct" abstract generalization of the zig-zag relation (this seems pretty hard). A simpler problem would be to provide a characterization of those rewriting systems were the zig-zag works well (or at least some sufficient conditions over the abstract systems). In other words, we should understand what are the "magical" aspects of the λ-calculus which allow such a simple definition of the family relation.

The same considerations also hold for the generalization of the extraction process to arbitrary IS's. Note that the extraction process is particularly interesting for Computer Science, since it essentially expresses the idea of *causality*.

Also in the case of labelling, a lot of interesting problems are still open. Our proof of the main theorem 8.1 is still very complex: any simplification would be welcome. Moreover, we have proved in this paper that there is a lot of redundancy in Lévy's labels (even after Klop's "simplification"). This suggests that it should be possible to define simpler labelling systems. In particular, it would be very interesting to exploit labelled calculi closer to the optimal implementation with Lamping's operators. A solution to this problem in the simple case of the λ-calculus would already be of great help to understand the general case of IS's.

11.4 Semantics

Well, there is nothing at all here, for the moment.

Acknowledgements: We would like to thank J.J.Lévy and G.Gonthier for the interesting discussions and their relevant suggestions.

This work is part of the Ph.D. Thesis of C. Laneve [Lan92], carried on under the supervision of U. Montanari.

Cosimo Laneve would like to thank the INRIA for its worm hospitality during his permanence in Spring '92.

References

[Ac78] P. Aczel *A General Church-Rosser Theorem.* Draft, Manchester, 1978.

[As91] A. Asperti *Linear Logic, Comonads and Optimal Reductions.* Draft, INRIA-Rocquencorut, 1991.

[Ba77] H. Barendregt *The type free lambda calculus.* In Handbook of Mathematical Logic (Barwise ed.), Studies in Logic 90. North Holland, Amsterdam. 1977.

[Be78] G. Berry *Séquentialitè de l'evaluation formelle des λ-expressions.* Proceedings of the 3-eme Colloque International sur la Programmation, Paris. 1978.

[Da90] V. Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation.* Thèse de doctorat, Univer sité Paris VII. 1990.

[Fi90] J. Field. *On laziness and optimality in lambda interpreters: tools for specification and analysis.* Proc. of the 17th Symposium on Principles of Programming Languages (POPL 90). San Francisco. 1990.

[Gi86] J. Y. Girard. *Linear Logic.* Theoretical Computer Scie nce, 50. 1986.

[Gi88] J. Y. Girard. *Geometry of Interaction I: Interpretation of system F.* In Ferro, Bonotto, Valentini and Zanardo eds., Logic Colloquium '88, North Holland. 1988.

[GAL92] G. Gonthier, M. Abadi, J.J. Lévy. *The geometry of optimal lambda reduction.* Proceedings of POPL'92.

[GAL92] G. Gonthier, M. Abadi, J.J. Lévy. *Linear Logic without boxes.* Proceedings of LICS'92.

[Ka90] V. Kathail *Optimal Interpreters for lambda-calculus based functional languages.* Ph.D Thesis, M.I.T. 1980.

[Kl80] J. W. Klop *Combinatory Reduction System.* Ph.D Thesis, Mathematisch Centrum, Amsterdam. 1980.

[Laf90] Y. Lafont. *Interaction Nets.* Proc. of the 17th Symposium on Principles of Programming Languages (POPL 90). San Francisco. 1990.

[Lam89] J. Lamping. *An algorithm for optimal lambda calculus reductions.* Techincal Report, Xerox PARC. 1989. on Principles of Programming Languages (POPL 90). San Francisco. 1990.

[Lam90] J. Lamping. *An algorithm for optimal lambda calculus reductions.* Proc. of the 17th Symposium on Principles of Programming Languages (POPL 90). San Francisco. 1990.

[Lan92] C. Laneve. *Concurrency and Optimality in Interaction Systems.* Ph.D. Thesis (in preparation), Dipartimento di Informatica, Pisa. 1992.

[Le78] J.J.Levy. *Réductions correctes et optimales dans le lambda-calcul.* Thèse de doctorat d'état, Université de Paris VII. 1978.

[Le80] J.J.Levy. *Optimal reductions in the lambda-calculus.* In J.P.Seldin and J.R.Hindley editors, "To H.B.Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism", Academic Press. 1980.

[St74] J. Staples. *Efficient Combinatory Reduction.* Queensland Institute of Techn., Math. dept., Brisbane, Australia. 1974.

[Vu74] J.Vuillemin. *Syntaxe, sémantique et axiomatique d'un langage de programmation simple.* Thèse de doctorat d'état, Université de Paris VII. 1974.

A The labelled calculus as an IS

In Section 5 we have turned *IS*'s into labelled variants which are left normal *CRS*. We devote this appendix to show a close simulation of such *CRS*'s through *IS*'s. As a consequence properties of the firsts can be proved by reasoning about the latters.

Let $(\Sigma,\ R)$ be an *IS* and $(\Sigma^L,\ R^L)$ its labelled variant of Section 5. The *IS* $(\Sigma^L_{IS},\ R^L_{IS})$ is such that Σ^L_{IS} consists of the following forms:

- if $\mathbf{d} \in \Sigma$ then $\mathbf{d} \in (\Sigma^L,\ R^L)$;
- if $\mathbf{c} \in \Sigma$ with arity n then, for every natural i, $\mathbf{c}_i \in \Sigma^L$ with arity $0^i n$;
- $\mathbf{l} \in \Sigma^L$ is a form of arity 00 and it is a destructor;
- if $\alpha \in \mathbf{L}_p$ then $\alpha \in \Sigma^L$ is a destructor of arity ε.

The set of rewriting rules R^L_{IS} is inductively generated as follows:

- for every rule $\mathbf{d}(\mathbf{c}(\vec{x}^{\,1}.X_1, \cdots, \vec{x}^{\,m}.X_m), \cdots, \vec{x}^{\,n}.X_n) \to H$ in R, for every i,

$$\mathbf{d}(\mathbf{c}_i(Y_1, \cdots, Y_i, \vec{x}^{\,1}.X_1, \cdots, \vec{x}^{\,m}.X_m), \cdots, \vec{x}^{\,n}.X_n) \to \mathcal{H}^0_{Y_1 \cdots Y_i}(H)$$

  is a rule in R^L_{IS};

- $\mathbf{l}(\mathbf{c}_i(Y_1, \cdots, Y_i, \vec{x}^{\,1}.X_1, \cdots, \vec{x}^{\,m}.X_m), Y_0) \to \mathbf{c}_{i+1}(Y_0, Y_1, \cdots, Y_i, \vec{x}^{\,1}.X_1, \cdots, \vec{x}^{\,m}.X_m)$ is a rule in R^L_{IS};

where the function $\mathcal{H}^0_X$ is defined over metaexpressions (of Σ) below:

$$\begin{aligned}
\mathcal{H}^s_X(x) &= \mathbf{l}(x, (X)_s) \\
\mathcal{H}^s_X(\mathbf{d}(H_0, \vec{x}_1.H_1, \cdots, \vec{x}_m.H_m)) &= \mathbf{l}(\mathbf{d}(\mathcal{H}^{s0}_X(H_0), \vec{x}^{\,1}.\mathcal{H}^{s1}_X(H_1), \cdots, \vec{x}^{\,m}.\mathcal{H}^{sm}_X(H_m)), (X)_s) \\
\mathcal{H}^s_X(\mathbf{c}(\vec{x}^{\,0}.H_0, \cdots, \vec{x}^{\,m}.H_m)) &= \mathbf{l}(\mathbf{c}_0(\vec{x}^{\,0}.\mathcal{H}^{s0}_X(H_0), \cdots, \vec{x}^{\,m}.\mathcal{H}^{sm}_X(H_m)), (X)_s) \\
\mathcal{H}^s_X(X[{}^{H_0}/_{x_0}, \cdots, {}^{H_m}/_{x_m}]) &= \mathbf{l}(X[{}^{\mathcal{H}^{s0}_X(H_0)}/_{x_0}, \cdots, {}^{\mathcal{H}^{sm}_X(H_m)}/_{x_m}], (X)_s)
\end{aligned}$$

We have this immediate fact:

Fact A.1 The rewriting system $(\Sigma^L_{IS}, R^L_{IS})$ is an *IS*. ∎

The close simulation of (Σ^L, R^L) by $(\Sigma^L_{IS}, R^L_{IS})$ is shown by providing an *embedding* function $\mathcal{E}$ between the two systems. Let $\mathcal{E}$ be such that:

$$(\Sigma^L, R^L) \xrightarrow{\mathcal{E}} (\Sigma^L_{IS}, R^L_{IS})$$

and defined inductively over expressions by:

$$\begin{aligned}
\mathcal{E}(x) &= x \\
\mathcal{E}(\alpha(t)) &= \mathbf{l}(\mathcal{E}(t), \alpha) \\
\mathcal{E}(\mathbf{d}(t_0, \vec{x}^1.t_1, \cdots, \vec{x}^m.t_m)) &= \mathbf{d}(\mathcal{E}(t_0), \vec{x}^1.\mathcal{E}(t_1), \cdots, \vec{x}^m.\mathcal{E}(t_m)) \\
\mathcal{E}(\mathbf{c}(\vec{x}^1.t_1, \cdots, \vec{x}^m.t_m)) &= \mathbf{c}_0(\vec{x}^1.\mathcal{E}(t_1), \cdots, \vec{x}^m.\mathcal{E}(t_m)))
\end{aligned}$$

It is immediate to show that $\mathcal{E}$ is injective. Here is the proposition stating the correctness of the implementation. Since $\mathcal{E}$ behaves homomorphically, for the sake of readability, we can safely restrict to expressions which match with lhs's of rewriting rules in R^L.

Proposition A.2 For every expression $t = \mathbf{d}(\alpha_1(\cdots(\alpha_i(\mathbf{c}(\vec{x}^1.t_1, \cdots, \vec{x}^m.t_m)\cdots), \cdots, \vec{x}^n.t_n))$ in Σ^L, $t \rightarrow t^+$ if and only if the following derivation

$$\begin{aligned}
\mathcal{E}(t) \quad &\rightarrow^j \quad \mathbf{d}(\mathbf{l}(\cdots\mathbf{l}(\mathbf{c}_j(\alpha_{i-j+1}, \cdots, \alpha_i, \vec{x}^1.\mathcal{E}(t_1), \cdots, \vec{x}^m.\mathcal{E}(t_m)), \alpha_{i-j}), \cdots), \alpha_1), \cdots, \vec{x}^n.\mathcal{E}(t_n)) \\
&\rightarrow^{i-j} \quad \mathbf{d}(\mathbf{c}_i(\alpha_1, \cdots, \alpha_i, \vec{x}^1.\mathcal{E}(t_1), \cdots, \vec{x}^m.\mathcal{E}(t_m)), \cdots, \vec{x}^n.\mathcal{E}(t_n)) \\
&\rightarrow \quad t'
\end{aligned}$$

is in $(\Sigma^L_{IS}, R^L_{IS})$.

Proof: The non trivial step is the last reduction. Observe that the existence of the last reduction in (Σ^L, R^L) or in $(\Sigma^L_{IS}, R^L_{IS})$ implies, by construction, the existence of the corresponding reduction in the other system. Thus it remains to check that $\mathcal{E}(t^+) = t'$.

We prove that if t' is the instance of $\mathcal{H}^s_\ell(H)$ then t^+ is the instance of $\mathcal{L}^s_\ell(H)$ and $\mathcal{E}(t^+) = t'$ ($\ell = \alpha_1 \cdots \alpha_i$). By induction over the structure H of the rhs of the rewriting rule contracting the pair **d**-**c** in (Σ, R).

(**case** $H = x$) $t' = \mathcal{H}^s_\ell(x) = \mathbf{l}(x, (\ell)_s) = \mathcal{E}((\ell)_s(x)) = \mathcal{E}(\mathcal{L}^s_\ell(x)) = \mathcal{E}(t^+)$.

(**case** $H = \mathbf{d}(H_0, \vec{x}^1.X_1, \cdots, \vec{x}^n.X_n)$) Then $t' = \mathbf{l}(\mathbf{d}(T_0, \vec{x}^1.T_1, \cdots, \vec{x}^n.T_n), (\ell)_s)$ and, by inductive hypothesis, $T_i = \mathcal{E}(T'_i)$ where T_i are the instances of $\mathcal{H}^{si}_\ell(H)$ and $\mathcal{L}^{si}_\ell(H)$, respectively. Thus $t' = \mathcal{E}((\ell)_s(\mathbf{d}(T'_0, \vec{x}^1.T'_1, \cdots, \vec{x}^n.T'_n)) = t^+$.

(**case** $H = \mathbf{c}(\vec{x}^0.X_0, \cdots, \vec{x}^n.X_n)$) Similar to the case of destructors.

(**case** $H = X_i[{}^{H_0}/_{x_0}, \cdots, {}^{H_n}/_{x_n}]$) Now $t' = \mathbf{l}(\mathcal{E}(t_i)[{}^{T_0}/_{x_0}, \cdots, {}^{T_n}/_{x_n}], (\ell)_s)$ where, by induction, $T_i = \mathcal{E}(T'_i)$ and T_i are the instances of $\mathcal{H}^{si}_\ell(H)$ and $\mathcal{L}^{si}_\ell(H)$, respectively. Therefore, it remains to prove:

$$\mathcal{E}(t_i)[{}^{T_0}/_{x_0}, \cdots, {}^{T_n}/_{x_n}] = \mathcal{E}(t_i[{}^{T'_0}/_{x_0}, \cdots, {}^{T'_n}/_{x_n}])$$

This equality follows by a simple structural induction over t_i. ∎

B About the structure of labels

This appendix is completely devoted to the proof of theorem B.1, that is

Theorem B.1 Let t be a term such that **INIT**(t), and let ℓ be a label generated along some labelled reduction σ. Then ℓ is uniquely determined by its structure **str**(ℓ) and a single atomic label of L in a specified occurrence inside **str**(ℓ). ■

As we already mentioned, the idea of the proof is that given an occurrence of an atomic symbol inside **str**(ℓ), we may uniquely derive all the other symbols "by connection".

Our first concern is to formalize this idea of connection.

We shall start with considering the "connections" of proper labels. Connections will be ports of specified instances of forms. In particular, every proper label α will have two connections, respectively denoted: α^- and α^+. Now, two cases are possible.

The first case is when α is atomic. Then it uniquely defines an edge in the original term t (we assume **INIT**(t)), and the two connections α^- and α^+ are the two connections of this edge.

The other case, is when the proper label α has been created by the firing of some rule of the labelled system. In this case, we have the following definition

Definition B.2 (Connections of a label) Given a labelled *IS* (Σ^L, R^L), let r^L be a rewriting rule in R^L and let

$$\mathbf{d}(\alpha_1(\cdots(\alpha_k(\mathbf{c}(\vec{x}^{\,1}.X_1,\cdots,\vec{x}^{\,n}.X_n)\cdots),\cdots,\vec{x}^{\,m}.X_m)$$

be its lhs. Let $\ell = \alpha_1 \cdots \alpha_k$, the *connections* β^+ and β^- of the proper label $\beta = (\ell)_s$ with respect to an instance of the rule r^L are defined by cases according to the position of β in the rhs of r^L:

β^- : There are three subcases:

1. in the situation $\beta(\mathbf{f}(\ldots))$, β^- is *connected to* the positive port of the output partition of the form **f**;
2. in the situation $\beta(x)$, where x is a bounded variable, β^- is *connected to* the suitable bounded port of the binder of x (note that according to the restrictions on IS-rules, this binder is eventually in the rhs of the instance of r^L);
3. in the situation $\beta(X_i[\ldots])$, where X_i is the i-th metavariable of **c** (**d**), β^- is *connected through* the unique negative port of the i-th input partition of **c** (resp. **d**). That is, the connection "passes through" the form **c** or **d** containing X (which is erased along the reduction). This is the reason of the apparent type mismatching: at the port expressing the connection, β^- will actually meet the positive port of the label contiguous to this port. Such a label β will be called a (negative) *border* label, since it (negatively) traverses the *border* between the rhs of the rule and a metavariables.

β^+ : Again, we shall distinguish several subases:

1. $(\ell)_0^+$ is *connected through* the output partition of the destructor **d**. As in the third case above, the connection of the label traverses the destructor and will meet the label connected to the output partition of **d**. This label opens the subterm relative to the rhs of r^L; it is a (positive) *border* label.
2. In the situation $\mathbf{f}(\ldots, \vec{x}^i.\beta(H_i), \ldots)$, β^+ is *connected to* the negative port of the i-th input partition of the form **f**.
3. In the situation $X_i[\ldots, {}^{\beta(H)}/_x, \ldots]$, where X_i is the i-th metavariable of **c** (resp. **d**), β^+ is *connected through* the positive bounded port of the i-th input partition of **c** (resp. **d**) individuated by x. In this case, the "control flow" comes back from the metavariable X_i to the rhs of the rule. Again, β is a (positive) *border* label. ■

Note that the previous definition is completely "local", i.e. it only concerns ports mentioned by the specified rule r^L, and no ports inside the metavariables.

Note also that a proper label may be at the same time a positive and negative border line; for instance this is always the case for not atomic proper labels in the λ-calculus.

Remark B.3 When the initial term is fully labelled, if a positive (negative) border label β appears inside a label generated along a derivation over some edge, we shall always find another label at its left (right). For this reason, instead of talking of positive (negative) border labels, we shall say sometimes, with some abuse of terminology, that a label is positively (negatively) *connected to another label.* Conversely, we will say that a proper label is positively (negatively) *connected to a form* when it is not a positive (negative) border label. ■

Definition B.4 (Consistent joins) Let α and β be two proper labels. The join $\alpha\beta$ is *consistent* if $\alpha^- = \beta^+$. ■

Proposition B.5 Rewriting rules make consistent joins.

Proof: Obvious, by definition of connection and by inspection of the rewriting rules. ■

Now we need a few properties concerning forms, and their "contiguous" labels. The idea is that proper labels contiguous to ports of a given form are never removed by reductions. The case of positive ports inside input-partitions (we will call them *bounded* ports), requires some care, since the number of links at these ports can change during the derivation, according to the number of bounded variables.

Definition B.6 Let **f** be a form in a term reached by a derivation σ from a term t such that **INIT**(t) holds.

1. **(Contiguos labels)** A label α is *contiguos* to **f** at a given port, if it is the *proper* label closest to **f** along the edge exiting from that port.
2. **(Interfaces)** The *interface* of **f** is the set of proper labels contiguous to *no bounded* ports of **f**. (We assume that the notion of interface also specifies, for every label, the port to which it is connected.) ■

The definition of residuals given in section 2 may be obviously extended to arbitrary forms. An alternative, simpler way is to mark a form $\mathbf{f}$ and to take as residuals of $\mathbf{f}$ along a derivation σ all the forms which are marked in the ending expression of σ. We have then some first simple results.

Lemma B.7 Interfaces are preserved by reductions. That is, if a form $\mathbf{f}'$ is a residual of $\mathbf{f}$ along a reduction σ, their interfaces are the same. Moreover, for every edge leading to a bounded port of $\mathbf{f}'$, if any, there exists an edge leading to the corresponding bounded port of $\mathbf{f}$ having the same contiguous label.

Proof: The proof is by induction over the length of σ. The base case is obvious. When $\sigma = \sigma'; \mathbf{u}$, assume that the property holds in the ending expression of σ'. By cases over the relative positions of u and $\mathbf{f}$.

(**case** u internal to the i-th argument of $\mathbf{f}$) According to the labelled rewriting rule contracting u, by no means the interface of $\mathbf{f}$ can be changed by σ. Notice that, if $\mathbf{f}$ binds at the i-th argument, it is possible that the edges incoming into a bounded port x can be deleted or copied. Assume that an edge incoming into the bounded port x of $\mathbf{f}'$ follows from copying. Then its contiguous label must be the same of the original edge copied in $\mathbf{f}$ because it must appear inside some metavariable of the rewriting schema and it is copied without breaking joins. Thus the thesis.

(**case** u external to $\mathbf{f}$) Then $\mathbf{f}$ must be inside one metavariable X of the rule contracting u. Thus, if $\mathbf{f}'$ is a residual of $\mathbf{f}$ along $\mathbf{u}$ then $\mathbf{f}'$ has still the same interface of $\mathbf{f}$ because X can be duplicated as a basic unit. Similarly, no change concerns edges incoming into bounded ports of $\mathbf{f}$.

(**case** σ is disjoint w.r.t. $\mathbf{f}$) Obvious. ■

Lemma B.8 Let $\mathbf{f}$ be a form in a term reached by a derivation σ from a term t such that **INIT**(t) holds. Then

1. all the labels in the interface of $\mathbf{f}$ are different from each other.
2. if a label contiguous to $\mathbf{f}$ is atomic then all the others are atomic, too.

Proof: (1) by the previous lemma, it suffices to prove the statement at the time $\mathbf{f}$ has been created. If $\mathbf{f}$ is a residual of a form in t, this is true by **INIT**(t). Otherwise, it is true by the definition of labelling in the rhs of the rewriting rules.

(2) According to the labelled rewriting rules, forms created along σ have not atomic contiguous labels. Therefore $\mathbf{f}$ is a residual of a form in t. The thesis is an obvious consequence of Lemma B.7. ■

Observe that, by Lemma B.8.(1), we have a bijective correspondence between ports of $\mathbf{f}$ which are not bounded and contiguos labels in the interface. Such correspondence fails for bounded ports, however the situation is similar. In particular, note that if $\mathbf{f}'$ is a residual of $\mathbf{f}$, we may have several contigouos labels to a same bounded port which are identical, due to duplication of bounded variables. However, by Lemma B.7 these contiguous labels are a subset of the initial ones. The converse is the interesting result: every label contiguous to a port of same form (the port can be bounded, too), uniquely identifies both the form and the bounded port (provided that **INIT** holds, of course).

Theorem B.9 Let **INIT**(t), and let α be a proper label contiguous to some form in an expression reached from t. Then α uniquely defines the form to which it is connected, the port where the connection takes place and the interface of the form.

Proof: By induction over the structure of the proper label α.

(**case** $\ell = a$, $a \in L$) Let $\mathbf{f}$ be the form connected to a^+ (resp. a^-) which must be determinated. Firstly observe that $\mathbf{f}$ must be residual of a form $\mathbf{f}'$ in t. Indeed, according to the labelled rewriting rules, forms created along derivations starting at t have contiguous labels that are not atomic. Therefore, the form $\mathbf{f}'$ can be determinated by looking at the form positively (resp. negatively) connected at the unique edge labelled a in t. If a is not connected to a bounded port, the theorem is an obvious consequence of Lemma B.7. If a^- is connected to a bounded port, again by Lemma B.7, there exists an edge incoming into the bounded port of $\mathbf{f}'$ having a as label. By **INIT**(t) there exists a unique node enjoying such property. The thesis is a consequence of Lemma B.7.

(**case** $\ell = (\ell')_s$) Note that the only possibility to create a label $(\ell')_s$ is to fire a redex whose degree is ℓ'. Let $\ell' = \alpha_1 \cdots \alpha_n$, where α_i's are proper. Thus the two forms connected to the edge labelled ℓ' must be a destructor and a constructor $\mathbf{d}$ - $\mathbf{c}$. Furthermore, these two forms, the ports involved by the connection and their interfaces only depend from α_1 and α_n. By induction there exists a unique pair of forms contiguous to α_1^+ and α_n^-. Therefore, it is possible to determinate the pair $\mathbf{d}$ - $\mathbf{c}$. Let r be the unique rewriting rule firing the pair. In order to discover the pair of forms contiguous to ℓ, we have just to look at the rhs of the labelled rule r. By definition there exists a unique occurrence of $(\ell')_s$ inside the rhs. Such occurrence must be positively (resp. negatively) connected to some form (otherwise the hypothesis should be invalidated). According to the definition of connection, the unique possibility is that this form is created by r. Thus we are able to determinate uniquely the port where ℓ is connected and the interface of the form by inspection of the rule r. By definition of connection, the unique case in which two forms may be connected by an edge labelled with $(-)_s$ is when they are both created by the rule. So the two forms connected by $(\ell')_s$ and the ports of this connection only depend from the rule r, which is uniquely individuated by ℓ'. ■

Corollary B.10 Let **INIT**(t), and let ℓ be a label of an edge generated along a derivation starting at t. Then ℓ uniquely defines the two forms connected by the edge marked by it, the two ports where the connection takes place and the interfaces of the forms.

Proof: Obvious consequence of Theorem B.9. ■

Let us come to the proof of the main theorem.

Proof of Theorem ??: We shall prove that, as soon as we know the structure of a label ℓ, any proper label α inside ℓ uniquely defines its "neighbours". As soon as we discover its neighbours, we reiterate the process, individuating in this way the entire label.

The proof is by cases according to the connections of α^- and α^+. Let us list them.

connection of α^-: (1) to the principal port of a constructor; **(2)** to the auxiliary port of the output partition of a destructor; **(3)** to a bounded port of a destructor; **(4)** to a bounded port of a

constructor; **(5)** to the positive port of a proper label (i.e. α is a negative border label, see remark B.3).

connection of α^+: (A) to the principal port of a destructor; **(B)** to an auxiliary port of a destructor; **(C)** to an auxiliary port of a constructor; **(D)** to the negative port of a proper label (i.e. α is a positive border label, see remark B.3.

(negative connection) If **(1)** then let **c** be the constructor to which α is connected. By definition of connection, the structure at its right (if any) must eventually be a bracket $)_s$, for some s. That is α is the innermost label marking a redex whose constructor is **c**. More precisely, we have a label as $\alpha_1 \cdots \alpha_n \alpha$ which marks a redex. As soon as $\alpha_n, \cdots, \alpha_1$ are discovered, we can apply Theorem B.9 in order to derive the connections of $(\alpha_1 \cdots \alpha_n \alpha)_s$ and, thus, reiterate the procedure.

When **(2)** holds, let **d** be the destructor connected to α^-. Then, if a structure is present to the right of α, it must be a label $(\alpha_1 \cdots \alpha_n)_0$ where α_1 is the label contiguous to the principal port of **d**. By Theorem B.9, α_1 can be determinated uniquely.

If we are in case **(3)**, the structure to the right of α is $(\alpha_1 \cdots \alpha_n)_s$ where, again, α_1 is the label contiguous to the principal port of the destructor connected to α^-. Theorem B.9 ensures that α_1 can be determinated uniquely. The situation is dual in case **(4)**, where we have a similar structure to the right but, now α_n is the label contiguous to the principal port of the constructor connected to α^-. Also α_n can be determinated uniquely. If **(5)**, then α cannot be atomic. Thus let $\alpha = (\ell)_s$. By definition of connection, the contiguous label to the right of α must be one (and it is determinated uniquely by the index s) of those contiguous to the negative, auxiliary ports of the constructor or destructor involved in the redex marked by ℓ, and by Theorem B.9, such labels can be uniquely determined. In particular, α is eventually a label of some metavariable X of the reduction marked by ℓ. Suppose X is the i-th metavariable of the constructor **c** (of the destructor **d**); then at the right of α we shall find the label contiguous to the unique negative port of the i-th input partition of **c** (**d**).

(positive connection) The case **(A)** is dual of **(1)**. In particular, here, the form connected to α^+ is a destructor **d**. By definition of connection, the structure at its right (if any) must eventually be an open bracket. That is α is the outermost label marking a redex whose destructor is **d**. Let $(\alpha\alpha_1 \cdots \alpha_n)_s$ be such label. As soon as we determinate $\alpha_1, \cdots, \alpha_n$, we can apply Theorem B.9 in order to derive the connections of $(\alpha_1 \cdots \alpha_n \alpha)_s$ and, thus, reiterate the procedure.

If **(B)** then, according to the definition of connection, to the left of α we must find a structure like $(\alpha_1, \cdots, \alpha_n)_s$, where α_1 is the proper label contiguous to the principal port of the destructor which is connected to α^+. Theorem B.9 guarantees that α_1 can be determinated uniquely. Case **(C)** is dual of the latter one. Here the proper label we are able to discover is α_n. The interesting case is **(D)**. Note that, till now, we never use the information borrowed by the indexes of bullets in the structure. Furthermore α must be structured. Let $\alpha = (\ell)_s$. The unique possibility to yield a label of this format is to fire a redex marked by ℓ. Thus, by Theorem B.9, ℓ defines a pair **d**-**c** (and, hence, the rewriting rule contracting such redex). Since α^+ is connected to one contiguous label of a bounded port x, then we must find the label relative to a *particular instance* of the variable x. Two cases are possible.

- At the left of α we may find the structure $\bullet_n$ (this is always the case, in the λ-calculus). Then this bullet represents a label contiguous to the form binding x (i.e. **c** or **d**) in the suitable port individuated by x. In the case of λ-calculus we do not even have the problem of individuating the variable, since abstraction is always over a single variable. By definition of structure, $\bullet_n$ represents an atomic label. Thus the form binding x must be a residual of a form **f** in the initial expression t (forms created by σ, have, by definitions of labelled rewriting rules, structured contiguous labels). Hence, by Lemma B.7, the labels contiguous to the bounded port x must be among those contiguous to the same bounded port of **f**. Now, by definition of structure, the index n uniquely determinates such atomic label.

- At the left of α we may find a structure $(\ell')_{s'}$. This means that the bounded variable has been created along the reduction, and it has been eventually created at the same time of its binder. Note that the binder is one of the two forms **c** and **d** involved in the reduction labelled α. Suppose the binder is **c** (the case of **d** is analogous). Recall that $\alpha = (\ell)_s$ and suppose that $\ell = \beta_1 \ldots \beta_k$, where every β_i is a proper label. Then β_k belongs to the interface of **c** (in the case of the destructor we shall take β_1). Since **c** has been created at the same time of x, β_k must be of the kind $(\ell')_{s''}$ (note the redundancy of information, here!). ■

ISSN 0249-6399